TMS320C645x Serial Rapid IO (SRIO)

User's Guide

Literature Number: SPRU976
March 2006

Texas Instruments

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document describes the Serial Rapid IO (SRIO) on the TMS320C645x devices.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.
- The term "word" describes a 32-bit value. The term "halfword" describes a 16-bit value.

Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

TMS320C6000 CPU and Instruction Set Reference Guide (literature number SPRU189) gives an introduction to the TMS320C62x™ and TMS320C67x™ DSPs, development tools, and third-party support.

TMS320C6455 Technical Reference (literature number SPRU965) gives an introduction to the TMS320C6455™ DSP and discusses the application areas that are enhanced.

TMS320C6000 Programmer's Guide (literature number SPRU198) describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

TMS320C6000 Code Composer Studio Tutorial (literature number SPRU301) introduces the Code Composer Studio™ integrated development environment and software tools.

Code Composer Studio Application Programming Interface Reference Guide (literature number SPRU321) describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

TMS320C64x+ Megamodule Reference Guide (literature number SPRU871) describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

TMS320C645x DSP Peripherals Overview Reference Guide (literature number SPRUE52) provides a brief description of the peripherals available on the TMS320C645x digital signal processors (DSPs).

TMS320C6455 Chip Support Libraries (CSL) (literature number SPRC234) is a download with the latest chip support libraries.

Trademarks

C6000, TMS320C62x, TMS320C67x, TMS320C6455, TMS320C6000, Code Composer Studio, RapidIO are trademarks of Texas Instruments.

InfiniBand is a trademark of the InfiniBand Trade Association.

Serial RapidIO (SRIO)

1 Overview

The RapidIO peripheral used in the TMS320C645x is called a serial RapidIO (SRIO). This chapter describes the general operation of a RapidIO system, how this module is connected to the outside world, the definitions of terms used within this document, and the features supported and not supported for SRIO.

1.1 General RapidIO System

RapidIO™ is a non-proprietary high-bandwidth system level interconnect. It is a packet-switched interconnect intended primarily as an intra-system interface for chip-to-chip and board-to-board communications at Gigabyte-per-second performance levels. Uses for the architecture can be found in connected microprocessors, memory, and memory mapped I/O devices that operate in networking equipment, memory subsystems, and general purpose computing. Principle features of RapidIO include:

- Flexible system architecture allowing peer-to-peer communication
- Robust communication with error detection features
- Frequency and port width scalability
- Operation that is not software intensive
- High bandwidth interconnect with low overhead
- Low pin count
- Low power
- Low latency

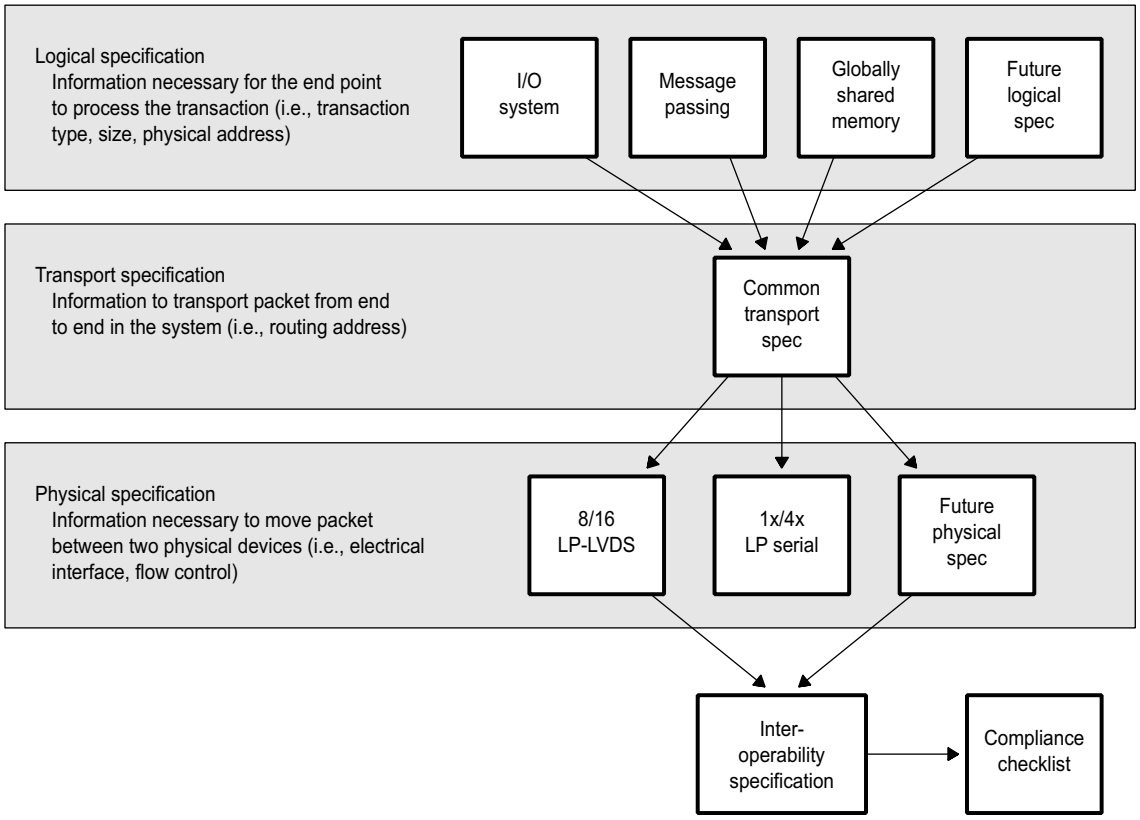
1.1.1 RapidIO Architectural Hierarchy

RapidIO is defined as a 3-layer architectural hierarchy.

- **Logical layer:** Specifies the protocols, including packet formats, which are needed by endpoints to process transactions
- **Transport layer:** Defines addressing schemes to correctly route information packets within a system
- **Physical layer:** Contains the device level interface information such as the electrical characteristics, error management data, and basic flow control data

In the RapidIO architecture, a single specification for the transport layer is compatible with differing specifications for the logical and physical layers (see Figure 1).

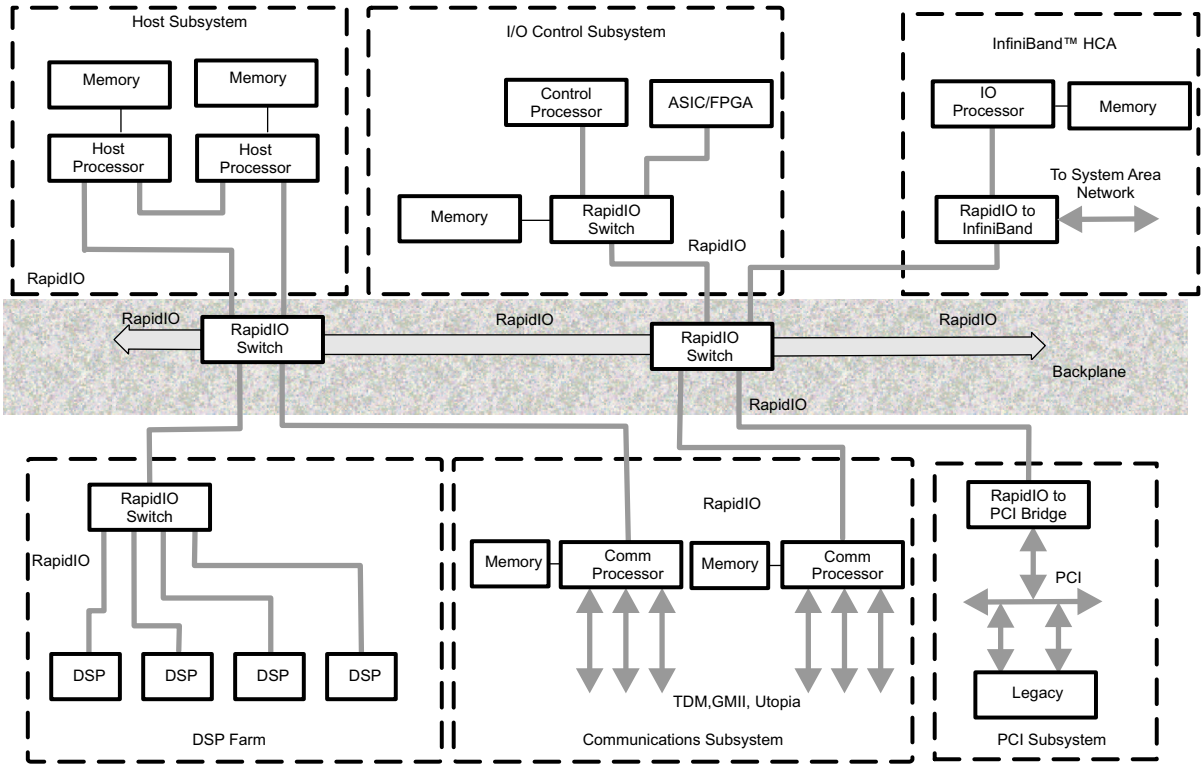
Figure 1. RapidIO Architectural Hierarchy



1.1.2 RapidIO Interconnect Architecture

The interconnect architecture is defined as a packet switched protocol independent of a physical layer implementation. Figure 2 illustrates the interconnection system.

Figure 2. RapidIO Interconnect Architecture



(1) InfiniBand™ is a trademark of the InfiniBand Trade Association.

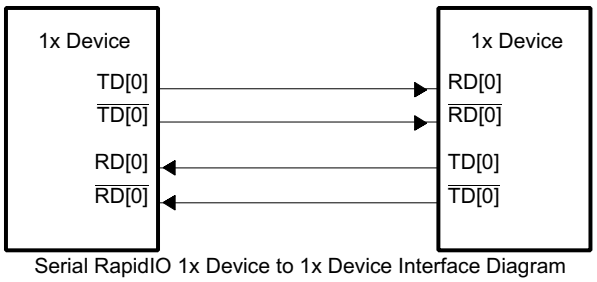
1.1.3 1x/4x LP-Serial

Currently, there are two physical layer specifications recognized by the RapidIO Trade Association: 8/16 LP-LVDS and 1X/4X LP-Serial. The 8/16 LP-LVDS specification is a point-to-point synchronous clock sourcing DDR interface. The 1X/4X LP-Serial specification is a point-to-point, AC coupled, clock recovery interface. The two physical layer specifications are not compatible.

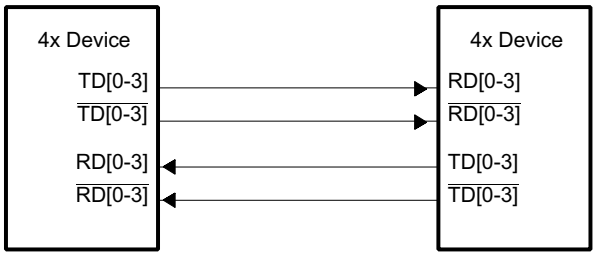
SRIO complies with the 1X/4X LP-Serial specification. The serializer/deserializer (SERDES) technology in SRIO also aligns with that specification.

The 1X/4X LP-Serial specification currently covers three frequency points: 1.25, 2.5, and 3.125 Gbps. This defines the total bandwidth of each differential pair of I/O signals. An 8b/10b encoding scheme ensures ample data transitions for the clock recovery circuits. Due to the 8b/10b encoding overhead, the effective data bandwidth per differential pair is 1.0, 2.0, and 2.5 Gbps respectively. Serial RapidIO only specifies these rates for both the 1X and 4X ports. A 1X port is defined as 1 TX and 1 RX differential pair. A 4X port is a combination of four of these pairs. This document describes a 4X RapidIO port that can also be configured as four 1X ports, thus providing a scalable interface capable of supporting a data bandwidth of 1 to 10 Gbps.

Figure 3. Serial RapidIO Device to Device Interface Diagrams



Serial RapidIO 1x Device to 1x Device Interface Diagram



Serial RapidIO 4x Device to 4x Device Interface Diagram

1.2 RapidIO Feature Support in SRIO

Features Supported in SRIO:

- RapidIO Interconnect Specification V1.2 compliance, Errata 1.2
- LP-Serial Specification V1.2 compliance
- 4X Serial RapidIO with auto-negotiation to 1X port, optional operation for four 1X ports
- Integrated clock recovery with TI SERDES
- Hardware error handling including Cyclic Redundancy Code (CRC)
- Differential CML signaling supporting AC and DC coupling
- Support for 1.25, 2.5, and 3.125 Gbps rates
- Power-down option for unused ports
- Read, write, write with response, streaming write, outgoing Atomic, and maintenance operations
- Generates interrupts to the CPU (Doorbell packets and internal scheduling)
- Support for 8b and 16b device ID
- Support for receiving 34b addresses
- Support for generating 34b, 50b, and 66b addresses
- Support for the following data sizes: byte, half-word, word, double-word
- Big endian data transfers
- Direct IO transfers
- Message passing transfers
- Data payloads of up to 256B
- Single messages consisting of up to 16 packets
- Elastic storage FIFOs for clock domain handoff
- Short run and long run compliance
- Support for Error Management Extensions
- Support for Congestion Control Extensions
- Support for one multi-cast ID

Features Not Supported:

- Compliance with the Global Shared Memory specification (GSM)
- 8/16 LP-LVDS compatible
- Destination support of RapidIO Atomic Operations
- Simultaneous mixing of frequencies between 1X ports (all ports must be the same frequency)
- Target atomic operations (including increment, decrement, test-and-swap, set, and clear) for internal L2 memory and registers

1.3 Standards

The SRIO peripheral is compliant to V1.2 of the RapidIO Interconnect Specification and V1.2 of the LP-Serial specification.

Table 1. *RapidIO* Documents and Links

Document	Link	Description
Official *RapidIO* Web Site	http://www.RapidIO.org	Various associated docs

1.4 External Devices Requirements

SRIO provides a seamless interface to all devices which are compliant to V1.2 of the LP-Serial RapidIO specification. This includes ASIC, microprocessor, DSP, and switch fabric devices from multiple vendors. Compliance to the specification can be verified with bus-functional models available through the RapidIO Trade Association, as well as test suites currently available for licensing.

2 SRIO Functional Description

2.1 Overview

2.1.1 Peripheral Data Flow

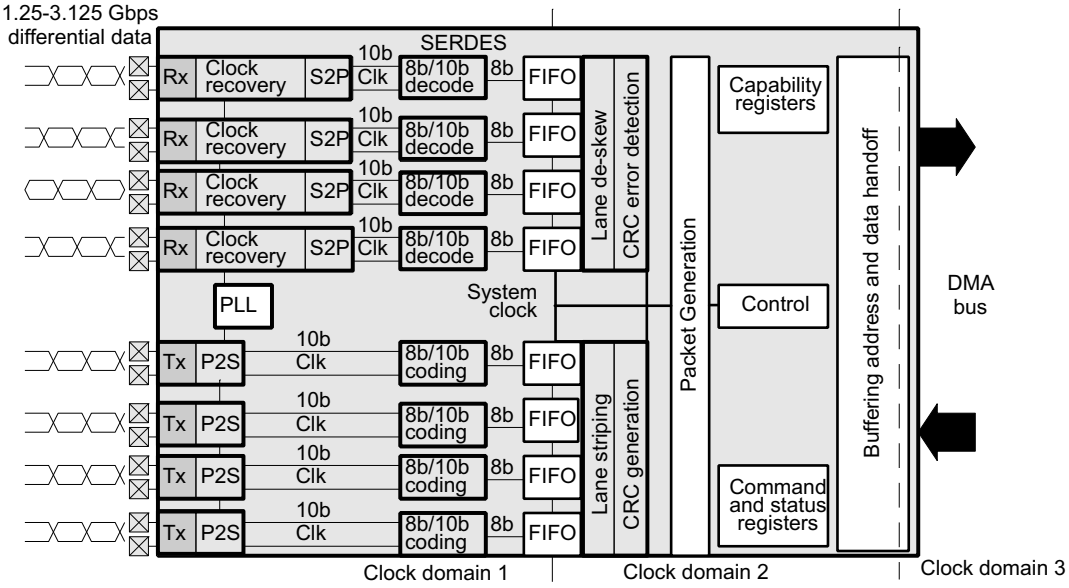
This peripheral is designed to be an external slave module that is capable of mastering the internal DMA. This means that an external device can push (burst write) data to the DSP as needed, without having to generate an interrupt to the CPU. This has two benefits. It cuts down on the total number of interrupts, and it reduces handshaking (latency) associated with read-only peripherals.

SRIO specifies data packets with payloads up to 256 bytes. Many times, transactions will span across multiple packets. RapidIO specifies a maximum of 16 transactions per message. Although a request is generated for each packet transaction so that the DMA can transfer the data to L2 memory, an interrupt is only generated after the final packet of the message. This interrupt notifies the CPU that data is available in L2 Memory for processing.

As an endpoint device, the peripheral accepts packets based on the destination ID. Two options exist for packet acceptance and are mode selectable. The first option is to only accept packets whose DestIDs match the local deviceID in 0x0080. This provides a level of security. The second option is to accept incoming packets matching the deviceID in either 0x0080 or 0x0084. This allows for system multicast operations.

Data flow through the peripheral can be explained using the high-level block diagram shown in Figure 4. High-speed data enters from the device pins into the RX block of the SERDES macro. The RX block is a differential receiver expecting a minimum of 175mV peak-to-peak differential input voltage (Vid). Level shifting is performed in the RX block, such that the output is single ended CMOS. The serial data is then fed to the SERDES clock recovery block. The sole purpose of this block is to extract a clock signal from the data stream. To do this, a low-frequency reference clock is required, 1/10th or ½0th the data rate. For example, for 3.125 Gbps data, a reference clock of 312.5Mhz or 156.25Mhz is needed. Typically, this clock comes from an off-chip stable crystal oscillator and is a LVDS device input separate to the SERDES. This clock is distributed to the SERDES PLL block which multiplies that frequency up to that of the data rate. Eight phases of this high-speed clock are created and routed to the clock recovery blocks. The clock recovery block further interpolates eight times between these clock phases. This provides clock edge resolution of 1/96th the Unit Interval (UI). The clock recovery block samples the incoming data and monitors the relative positions of the data edges. With this information, it can provide the data and a center-aligned clock to the S2P block. The S2P block uses the newly recovered clock to demux the data into 10-bit words. At this point, the data leaves the SERDES macro at 1/10th the pin data rate, accompanied by an aligned byte clock.

Figure 4. SRIO Peripheral Block Diagram



Within the physical layer, the data next goes to the 8b/10b decode block. 8b/10b encoding is used by RapidIO to ensure adequate data transitions for the clock recovery circuits. Here the 20% encoding overhead is removed as the 10-bit data is decoded to the raw 8-bit data. At this point, the recovered byte clock is still being used.

The next step is clock synchronization and data alignment. These functions are handled by the FIFO and lane de-skewing blocks. The FIFO provides an elastic store mechanism used to hand off between the recovered clock domains and a common system clock. After the FIFO, the four lanes are synchronized in frequency and phase, whether 1X or 4X mode is being used. The FIFO is 8 words deep. The lane de-skew is only meaningful in the 4X mode, where it aligns each channel's word boundaries, such that the resulting 32-bit word is correctly aligned.

The CRC error detection block keeps a running tally of the incoming data and computes the expected CRC value for the 1X or 4X mode. The expected value is compared against the CRC value at the end of the received packet.

After the packet reaches the logical layer, the packet fields are decoded and the payload is buffered. Depending on the type of received packet, the packet routing is handled by functional blocks which control the DMA access.

2.1.2 SRIO Packets

The SRIO data stream consists of data fields pertaining to the logical layer, the transport layer, and the physical layer.

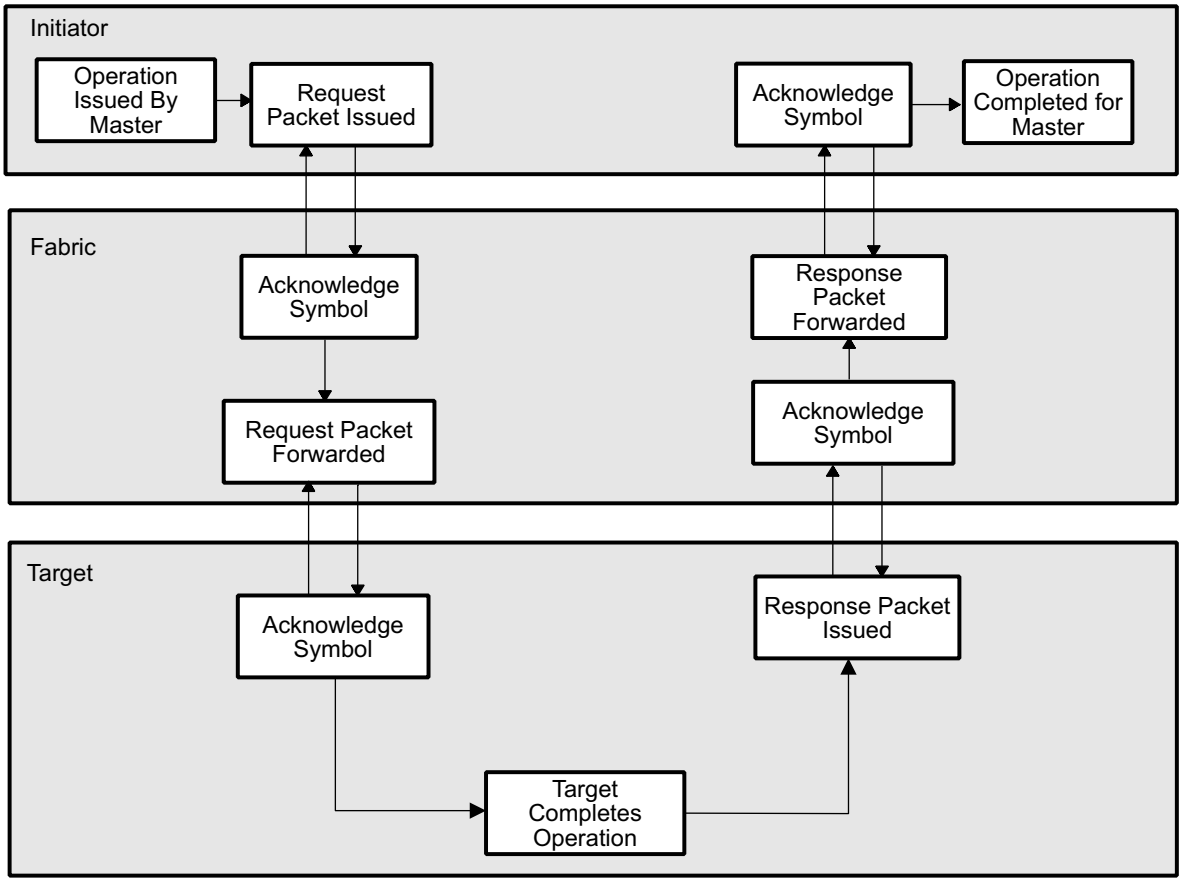
- The logical layer consists of the header (defining the type of access) and the payload (if present).
- The transport layer is partially dependent on the physical topology in the system, and consists of source and destination IDs for the sending and receiving devices.
- The physical layer is dependent on the physical interface (i.e., serial versus parallel RapidIO) and includes priority, acknowledgment, and error checking fields.

2.1.2.1 Operation Sequence

SRIO transactions are based on request and response packets. Packets are the communication element between endpoint devices in the system. A master or initiator generates a request packet which is transmitted to a target. The target then generates a response packet back to the initiator to complete the transaction.

SRIO endpoints are typically not connected directly to each other but instead have intervening connection fabric devices. Control symbols are used to manage the flow of transactions in the SRIO physical interconnect. Control symbols are used for packet acknowledgment, flow control information, and maintenance functions. Figure 5 shows how a packet progresses through the system.

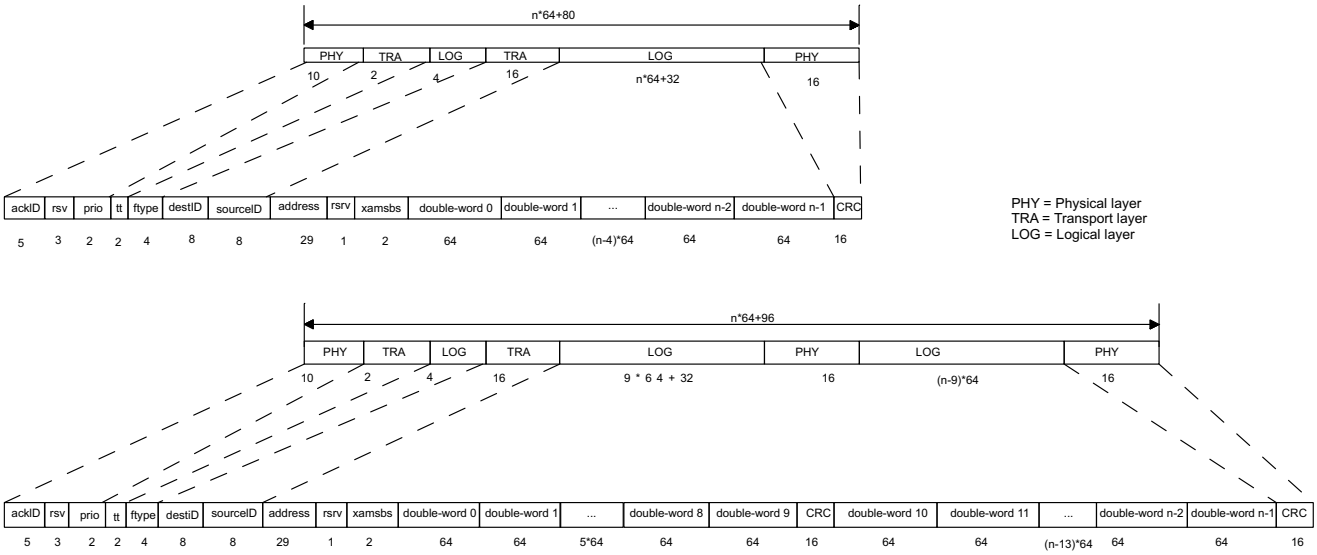
Figure 5. Operation Sequence



2.1.2.2 Example Packet – Streaming Write

An example packet is shown as two data streams in Figure 6. The first is for payload sizes of 80 bytes or less, while the second applies to payload sizes of 80 to 256 bytes. SRIO packets must have a length that is an even integer of 32 bits. If the combination of physical, logical and transport layers has a length that is an integer of 16 bits, a 16-bit pad of value 0x0000 is added to the end of the packet, after the CRC (not shown). Bit fields that are defined as reserved are assigned to logic 0s when generated and ignored when received. All request and response packet formats are described in the I/O and Message Passing Logical Specifications.

Figure 6. 1x/4x RapidIO Packet Data Stream (Streaming-Write Class)



Note: Figure 6 assumes that addresses are 32-bit and device IDs are 8-bit.

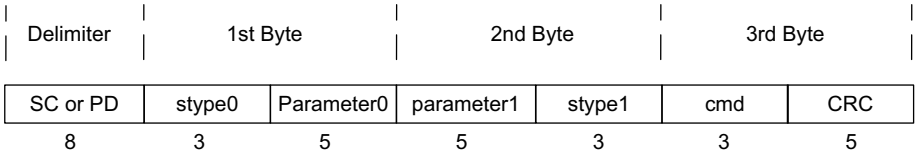
The device ID, being an 8-bit field, will address up to 256 nodes in the system. If 16-bit addresses were used, the system could accommodate up to 64k nodes.

The data stream includes a Cyclic Redundancy Code (CRC) field to ensure the data was correctly received. The CRC value protects the entire packet except the ackID and one bit of the reserved PHY field. The peripheral checks the CRC automatically in hardware. If the CRC is correct, a Packet-Accepted control symbol is sent by the receiving device. If the CRC is incorrect, a Packet-Not-Accepted control symbol is sent so that transmission may be retried.

2.1.2.3 Control Symbols

Control symbols are physical layer message elements used to manage link maintenance, packet delimiting, packet acknowledgment, error reporting, and error recovery. All transmitted data packets are delimited by start-of-packet and end-of-packet delimiters. SRIO control symbols are 24 bits long and are protected by their own CRC. Control symbols provide two functions: stype0 symbols convey the status of the port transmitting the symbol, and stype1 symbols are requests to the receiving port or transmission delimiters. They have the following format, which is detailed in section 3 of the RapidIO LP-Serial specification.

Figure 7. Serial RapidIO Control Symbol Format



Control symbols are delimited by special characters at the beginning of the symbol. If the control symbol contains a packet delimiter(start-of-packet, end-of-packet, etc.), the special character PD (K28.3) is used. If the control symbol does not contain a packet delimiter, the special character SC (K28.0) is used. This use of special characters provides an early warning of the contents of the control symbol. The CRC does not protect the special characters, but an illegal or invalid character is recognized and flagged as Packet-Not-Accepted. Since control symbols are known length, they do not need end delimiters.

The type of received packet determines how the packet routing is handled. Reserved or undefined packet types are destroyed before being processed by the logical layer functional blocks. This prevents erroneous allocation of resources to them. Unsupported packet types are responded to with an error response packet. Section 2.1.2.4 details the handling of such packets.

2.1.2.4 SRIO Packet Ftype/Ttype

The type of SRIO packet is determined by the combination of Ftype and Ttype fields in the packet. Table 2 lists all supported combinations of Ftype/Ttype and the corresponding decoded actions on the packets.

Table 2. Packet Type

Ftype	Ttype	Packet Type
Ftype=0,	Ttype=don't care	
Ftype=2,	Ttype=0100b,	NREAD
	Ttype=1100b,	Atomic inc
	Ttype=1101b,	Atomic dec
	Ttype=1110b,	Atomic set
	Ttype=1111b,	Atomic clr
	Ttype=others,	
Ftype=5,	Ttype=0100b,	NWRITE
	Ttype=0101b,	NWRITE_R
	Ttype=1110b,	Atomic t&s
	Ttype=others,	
Ftype=6,	Ttype=don't care,	SWRITE
Ftype=7,	Ttype=don't care,	Congestion
Ftype=8,	Ttype=0000b,	Mtn Rd
	Ttype=0001b,	Mtn Wr
	Ttype=0010b,	Mtn Rd Resp
	Ttype=0011b,	Mtn Wr Resp
	Ttype=0100b,	Mtn Pt-Wr
	Ttype=others,	
Ftype=10,	Ttype=don't care,	Doorbell
Ftype=11,	Ttype=don't care,	Message
Ftype=13,	Ttype=0000b,	Resp(+Dbll Resp)
	Ttype=0001b,	Message Resp
	Ttype=1000b,	Resp w/payload
	Ttype=other,	
Undefined Ftype (1,3,4,9,12,14,15):		

Packet type definition:

```
#define REQ_MAINT_RD  0x80 //0b10000000  // ftype=8
#define REQ_MAINT_WR   0x81 //0b10000001  // ftype=8
#define REQ_MAINT_PW   0x84 //0b10000100  // ftype=8
#define REQ_ATOMIC_INC 0x2C //0b00101100  // ftype=2
#define REQ_ATOMIC_DEC 0x2D //0b00101101  // ftype=2
#define REQ_ATOMIC_SET 0x2E //0b00101110  // ftype=2
#define REQ_ATOMIC_CLR 0x2F //0b00101111  // ftype=2
#define REQ_ATOMIC_TNS 0x5E //0b01011110  // ftype=5
#define REQ_NREAD      0x24 //0b00100100  // ftype=2
#define REQ_NWRITE     0x54 //0b01010100  // ftype=5
#define REQ_NWRITE_R   0x55 //0b01010101  // ftype=5
#define REQ_SWRITE     0x60 //0b01100000  // ftype=6
#define REQ_DOORBELL   0xA0 //0b10100000  // ftype=10
```

2.2 SRIO Pins

The SRIO device pins are high-speed differential signals based on Current-Mode Logic (CML) switching levels. The transmit and receive buffers are self-contained within the clock recovery blocks. The reference clock input is not incorporated into the SERDES macro. It uses a common LVDS input buffer that aligns interfaces with crystal oscillator manufacturers. None of the peripheral pins may be used as GPIO pins. Table 3 provides more detail.

Table 3. Pin Description

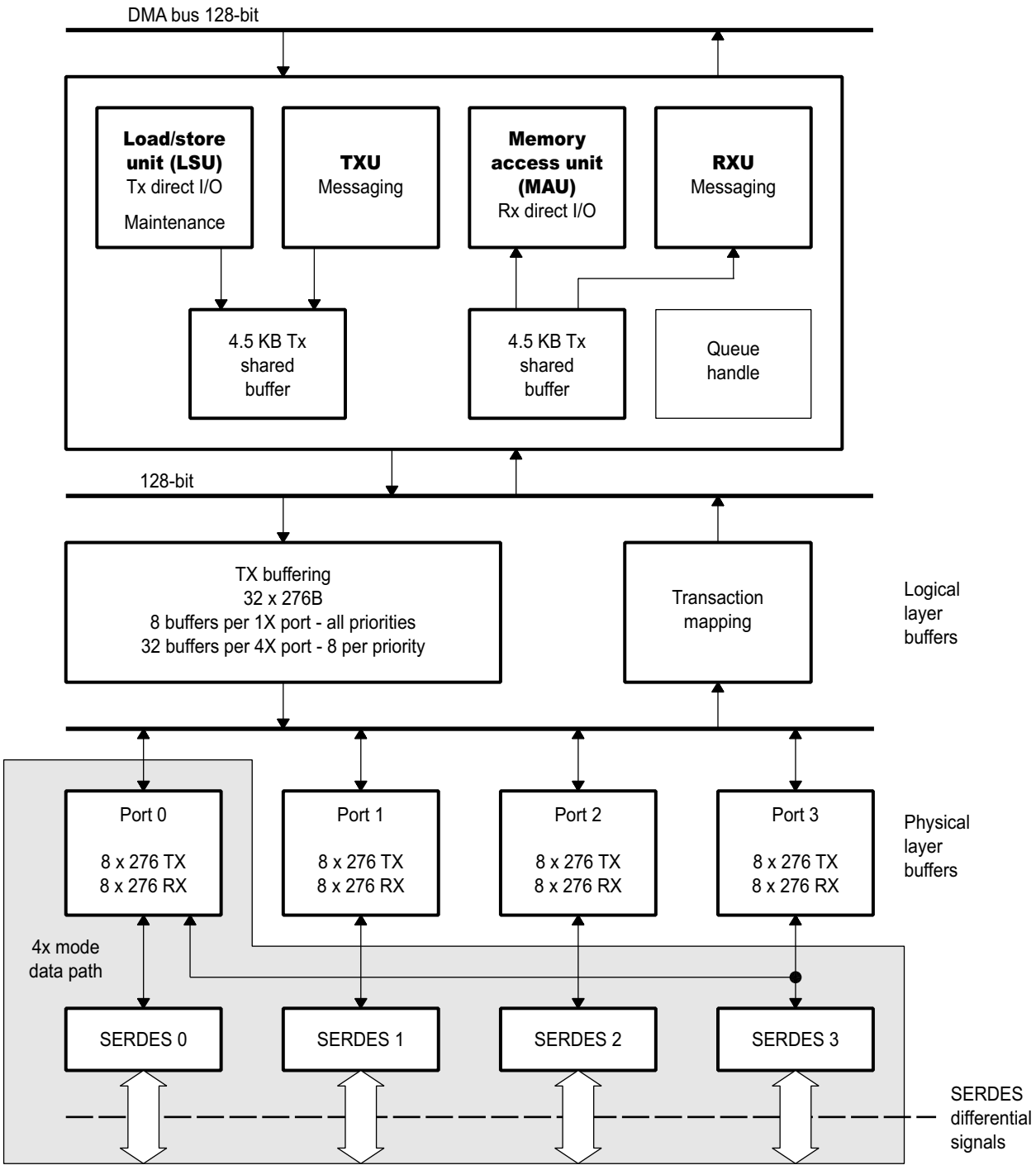
Pin Name	Pin Count	Signal Direction	Description
RIOTX3/ $\overline{\text{RIOTX3}}$	2	Output	Transmit Data – Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Most significant bit of a 4X device.
RIOTX2/ $\overline{\text{RIOTX2}}$	2	Output	Transmit Data – Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins.
RIOTX1/ $\overline{\text{RIOTX1}}$	2	Output	Transmit Data – Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins.
RIOTX0/ $\overline{\text{RIOTX0}}$	2	Output	Transmit Data – Differential point-to-point unidirectional bus. Transmits packet data to a receiving device's RX pins. Bit used for 1X mode.
RIORX3/ $\overline{\text{RIORX3}}$	2	Input	Receive Data – Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Most significant bits in 4X mode.
RIORX2/ $\overline{\text{RIORX2}}$	2	Input	Receive Data – Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins.
RIORX1/ $\overline{\text{RIORX1}}$	2	Input	Receive Data – Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins.
RIORX0/ $\overline{\text{RIORX0}}$	2	Input	Receive Data – Differential point-to-point unidirectional bus. Receives packet data for a transmitting device's TX pins. Bit used for 1X mode
RIOCLK/ $\overline{\text{RIOCLK}}$	2	Input	Reference Clock Input Buffer for peripheral clock recovery circuitry.

2.3 Functional Operation

2.3.1 Block Diagram

Figure 8 shows a conceptual block diagram of the SRIO peripheral. The load/store unit (LSU) controls the transmission of Direct I/O packets, and the memory access unit (MAU) controls the reception of Direct I/O packets. The LSU also controls the transmission of maintenance packets. Message packets are transmitted by the TXU and received by the RXU. These four units use the internal DMA to communicate with internal memory, and they use buffers and receive/transmit ports to communicate with external devices. Serializer/deserializer (SERDES) macros support the ports by performing the parallel-to-serial coding for transmission and serial-to-parallel decoding for reception.

Figure 8. SRIO Conceptual Block Diagram



2.3.2 SERDES and its Configurations

SRIO offers many benefits to customers by allowing a scalable non-proprietary interface. With the use of TI's SERDES macros, the peripheral is very adaptable and bandwidth scalable. The same peripheral can be used for all three frequency nodes specified in V1.2 of the RapidIO specification (1.25, 2.5, and 3.125 Gbps). This allows you to design to only one protocol throughout the system and selectively choose the bandwidth, thus eliminating the need for user's proprietary protocols in many instances, and providing a faster design turn and production ramp. Since this interface is serial, the application space is not limited to a single board. It will propagate into backplane applications as well. Integration of these macros on an ASIC or DSP allows you to reduce the number of discrete components on the board and eliminates the need for bus driver chips.

Additionally, there are some valuable features built into TI SERDES. System optimization can be uniquely managed to meet individual customer applications. For example, control registers within the SERDES allow you to adjust the TX differential output voltage (Vod) on a per driver basis. This allows power savings on short trace links (on the same board) by reducing the TX swing. Similarly, data edge rates can be adjusted through the control registers to help reduce any EMI affects. Unused links can be individually powered down without affecting the working links.

Because the high-speed analog nature of the SERDES is often the most critical portion of the RapidIO peripheral, good test access is important.

The SERDES is a self-contained macro which includes transmitter (TX), receiver (RX), phase-locked-loop (PLL), clock recovery, serial-to-parallel (S2P), and parallel-to-serial (P2S) blocks. The internal PLL multiplies a user-supplied reference clock. All loop filter components of the PLL are onchip. Likewise, the differential TX and RX buffers contain on-chip termination resistors. The only off-chip component requirement is for DC blocking capacitors. These capacitors are needed only to ensure interoperability between vendors and can be removed in cases where TI devices talk to other TI devices at the same voltage node. The SERDES are designed for 1.2V ± 5% operation. This provides for excellent power efficiency.

2.3.2.1 Enabling the PLL

The Physical layer SERDES has a built-in PLL, which is used for the clock recovery circuitry. The PLL is responsible for clock multiplication of a slow speed reference clock. This reference clock has no timing relationship to the serial data and is asynchronous to any CPU system clock. The multiplied high-speed clock is only routed within the SERDES block, it is not distributed to the remaining blocks of the peripheral, nor is it a boundary signal to the core of the device. It is extremely important to have a good quality reference clock, and to isolate it and the PLL from all noise sources. A unique Reference Clock Distribution (RCD) macro is used for this purpose. Since RapidIO requires 8b/10b encoded data, the 8-bit mode of the SERDES PLL will not be used.

SERDES_CFG*n*_CNTL, SERDES_CFGRX*n*_CNTL, and SERDES_CFGTX*n*_CNTL registers are used to configure SERDES. To enable the internal PLL, the ENPLL bit of SERDES_CFG*n*_CNTL must be set high. After setting this bit, it is necessary to allow 1µs for the regulator to stabilize. Thereafter, the PLL will take no longer than 200 reference clock cycles to lock to the required frequency, provided RIOCLK and $\overline{\text{RIOCLK}}$ are stable.

Table 4. Bits of SERDES_CFG*n*_CNTL Register (0x120 - 0x12c)

Bit	Name	Value	Description
31:10	Reserved		Reserved.
9:8	LB		Loop bandwidth. Specify loop bandwidth settings.
		00	Frequency dependent bandwidth. The PLL bandwidth is set to a twelfth of the frequency of RIOCLK and $\overline{\text{RIOCLK}}$.
		01	Reserved
		10	Low bandwidth. The PLL bandwidth is set to a twentieth of the frequency of RIOCLK and $\overline{\text{RIOCLK}}$, or 3MHz (whichever is larger).
		11	High bandwidth. The PLL bandwidth is set to an eighth of the frequency of RIOCLK and $\overline{\text{RIOCLK}}$.
7:6	Reserved		Reserved.

Table 4. Bits of SERDES_CFG*n*_CNTL Register (0x120 - 0x12c) (continued)

Bit	Name	Value	Description
5:1	MPY		PLL multiply. Select PLL multiply factors between 4 and 60. Multiply modes shown below.
		0000	4x
		0001	5x
		0010	6x
		0011	Reserved
		0100	8x
		0101	10x
		0110	12x
		0111	12.5x
		1000	15x
		1001	20x
		1010	25x
		1011	Reserved
		1100	Reserved
		1101	50x
		1110	60x
		1111	Reserved
0	ENPLL		Enable PLL. Enables the PLL.

Based on the MPY value, the following line rate versus PLL output clock frequency can be estimated:

Table 5. Line Rate versus PLL Output Clock Frequency

Rate	Line Rate	PLL Output Frequency	RATESCALE
Full	x Gbps	0.5x GHz	0.5
Half	x Gbps	x GHz	1
Quarter	x Gbps	2x GHz	2

$$\text{RIOCLK and } \overline{\text{RIOCLK}}_{FREQ} = \frac{LINERATE \times RATESCALE}{MPY}$$

The rate is defined by the RATE bits of the SERDES_CFGRX*n*_CNTL register and the SERDES_CFGTX*n*_CNTL register, respectively.

The primary operating frequency of the SERDES macro is determined by the reference clock frequency and PLL multiplication factor. However, to support lower frequency applications, each receiver and transmitter can also be configured to operate at a half or quarter of this rate via the RATE bits of the SERDES_CFGRX*n*_CNTL and SERDES_CFGTX*n*_CNTL registers as described in Table 6.

Table 6. RATE Bit Effects

Value	Description
0 0	Full rate. Two data samples taken per PLL output clock cycle.
0 1	Half rate. One data sample taken per PLL output clock cycle.
1 0	Quarter rate. One data sample taken every two PLL output clock cycles.
1 1	Reserved.

Here is the frequency range versus MPY:

Table 7. Frequency Range versus MPY

MPY	RIOCLK and $\overline{\text{RIOCLK}}$ Range (MHz)	Line Rate Range (Gbps)		
		Full	Half	Quarter
4x	250 - 425	2 - 3.4	1 - 1.7	0.5 - 0.85
5x	200 - 425	2 - 4.25	1 - 2.125	0.5 - 1.0625
6x	167 - 354.167	2 - 4.25	1 - 2.125	0.5 - 1.0625
8x	125 - 265.625	2 - 4.25	1 - 2.125	0.5 - 1.0625
10x	100 - 212.5	2 - 4.25	1 - 2.125	0.5 - 1.0625
12x	83.33 - 177.08	2 - 4.25	1 - 2.125	0.5 - 1.0625
12.5x	80 - 170	2 - 4.25	1 - 2.125	0.5 - 1.0625
15x	66.67 - 141.67	2 - 4.25	1 - 2.125	0.5 - 1.0625
20x	50 - 106.25	2 - 4.25	1 - 2.125	0.5 - 1.0625
25x	40 - 85	2 - 4.25	1 - 2.125	0.5 - 1.0625
50x	25 - 42.5	2.5 - 4.25	1.25 - 2.125	0.625 - 1.0625
60x	25 - 35.42	3 - 4.25	1.5 - 2.125	0.75 - 1.0625

2.3.2.2 Enabling the Receiver

To enable a receiver for deserialization, the ENRX bit of the associated SERDES_CFGRX*n*_CNTL registers (0x100-0x10c) must be set high.

When ENRX is low, all digital circuitry within the receiver will be disabled, and clocks will be gated off. All current sources within the receiver will be fully powered down, with the exception of those associated with the loss of signal detector and IEEE1149.6 boundary scan comparators. Loss of signal power down is independently controlled via the LOS bits of SERDES_CFGRX*n*_CNTL.

Table 8. Bits of SERDES_CFGRX*n*_CNTL Registers

Bit	Field	Value	Description
31:26	Reserved		Reserved.
25	Reserved		Reserved, keep as zero during writes to this register.
24	Reserved		Reserved, keep as zero during writes to this register.
23	Reserved		Reserved.
22:19	EQ		Equalizer. Enables and configures the adaptive equalizer to compensate for loss in the transmission media. For values, see Table 9.
18:16	CDR		Clock/data recovery. Configures the clock/data recovery algorithm.
		000	First order. Phase offset tracking up to ±488 ppm.
		001	Second order. Highest precision frequency offset matching but poorest response to changes in frequency offset, and longest lock time. Suitable for use in systems with fixed frequency offset.
		010	Second order. Medium precision frequency offset matching, frequency offset change response and lock time.
		011	Second order. Best response to changes in frequency offset and fastest lock time, but lowest precision frequency offset matching. Suitable for use in systems with spread spectrum clocking.
		100	First order with fast lock. Phase offset tracking up to ±1953 ppm in the presence of ..10101010.. training pattern, and ±448 ppm otherwise.
		101	Second order with fast lock. As per setting 001, but with improved response to changes in frequency offset when not close to lock.
		110	Second order with fast lock. As per setting 010, but with improved response to changes in frequency offset when not close to lock.
		111	Second order with fast lock. As per setting 011, but with improved response to changes in frequency offset when not close to lock.

Table 8. Bits of SERDES_CFGRX*n*_CNTL Registers (continued)

Bit	Field	Value	Description
15:14	LOS		Loss of signal. Enables loss of signal detection with 2 selectable thresholds.
		00	Disabled. Loss of signal detection disabled.
		01	High threshold. Loss of signal detection threshold in the range 85 to 195mV$_{dfpp}$. This setting is suitable for Infiniband.
		10	Low threshold. Loss of signal detection threshold in the range 65-175mV$_{dfpp}$. This setting is suitable for PCI-E and S-ATA.
		11	Reserved
13:12	ALIGN		Symbol alignment. Enables internal or external symbol alignment.
		00	Alignment disabled. No symbol alignment will be performed while this setting is selected, or when switching to this selection from another.
		01	Comma alignment enabled. Symbol alignment will be performed whenever a misaligned comma symbol is received.
		10	Alignment jog. The symbol alignment will be adjusted by one bit position when this mode is selected (i.e., CFGRX[13:12] changes from 0x to 1x).
		11	Reserved
11	Reserved		Reserved.
10:8	TERM		Termination. Selects input termination options suitable for a variety of AC or DC coupled scenarios.
		000	Common point connected to VDDT. This configuration is for DC coupled systems using CML transmitters. The common mode voltage is determined jointly by both the receiver and the transmitter. Common mode termination is via a 50 pF capacitor to VSSA.
		001	Common point set to 0.8 VDDT. This configuration is for AC coupled systems using CML transmitters. The transmitter has no effect on the receiver common mode, which is set to optimize the input sensitivity of the receiver. Common mode termination is via a 50 pF capacitor to VSSA.
		010	Reserved
		011	Common point floating. This configuration is for DC coupled systems that require the common mode voltage to be determined by the transmitter only. These are typically not CML. Common mode termination is via a 50 pF capacitor to VSSA.
		1xx	Reserved
7	INVPAIR		Invert polarity. Inverts polarity of RXP*n* and RXN*n*.
		0	Normal polarity. RXP*n* considered to be positive data and RXN*n* negative.
		1	Inverted polarity. RXP*n* considered to be negative data and RXN*n* positive.
6:5	RATE		Operating rate. Selects full, half or quarter rate operation.
		00	Full rate. Two data samples taken per PLL output clock cycle.
		01	Half rate. One data sample taken per PLL output clock cycle.
		10	Quarter rate. One data sample taken every two PLL output clock cycles.
		11	Reserved
4:2	BUS-WIDTH		Bus width. Selects the width of the parallel interface (10 or 8 bit).
		000	10-bit operation. Data is output on RD*n*[9:0]. RXBCLK[*n*] period is 10 bit periods (4 high, 6 low).
		001	8-bit operation. Data is output on RD*n*[7:0]. RXBCLK[*n*] period is 8 bit periods (4 high, 4 low). RD*n*[9:8] will replicate bits [1:0] from the previous byte.
		01x	Reserved
		1xx	Reserved
1	Reserved		Reserved, keep as zero during writes to this register.
0	ENRX		Enable receiver. Enables this receiver when high.
		0	Disable
		1	Enable

Table 9. EQ Bits

CFGRX[22:19]	Low Freq Gain	Zero Freq (at e_{28} (min))
0000	Maximum	-
0001	Adaptive	Adaptive
001x	Reserved	
01xx	Reserved	
1000	Adaptive	1084MHz
1001		805MHz
1010		573MHz
1011		402MHz
1100		304MHz
1101		216MHz
1110		156MHz
1111		135MHz

2.3.2.3 Enabling the Transmitter

To enable a transmitter for serialization, the ENTX bit of the associated SERDES_CFGTX*n*_CNTL registers (0x110 – 0x10c) must be set high. When ENTX is low, all digital circuitry within the transmitter will be disabled, and clocks will be gated off, with the exception of the transmit clock (TXBCLK[*n*]) output, which will continue to operate normally. All current sources within the transmitter will be fully powered down, with the exception of the CML driver, which will remain powered up if boundary scan is selected.

Table 10. Bits of SERDES_CFGTX*n*_CNTL Registers

Bit	Field	Value	Description
31:17	Reserved		Reserved.
16	ENFTP		Enable fixed TXBCLKIN*n* phase. Enables fixed phase relationship of TXBCLKIN*n* with respect to TXBCLK*n*.
		0	Arbitrary phase. No required phase relationship between TXBCLKIN*n* and TXBCLK*n*.
		1	Fixed phase. Requires direct connection of TXBCLK*n* to TXBCLKIN*n* using a minimum length net without buffers.
15:12	DE		De-emphasis. Selects one of 15 output de-emphasis settings from 4.76 to 71.42%. See Table 12.
11:9	SWING		Output swing. Selects one of 8 output amplitude settings between 125 and 1250mV_{dfpp}. See Table 11.
8	CM		Common mode. Adjusts the common mode to suit the termination at the attached receiver.
		0	Normal common mode. Common mode not adjusted.
		1	Raised common mode. Common mode raised by 5% of e_{54}.
7	INVPAIR		Invert polarity. Inverts polarity of TXP*n* and TXN*n*.
		0	Normal polarity. TXP*n* considered to be positive data and TXN*n* negative.
		1	Inverted polarity. TXP*n* considered to be negative data and TXN*n* positive.
6:5	RATE		Operating rate. Selects full, half or quarter rate operation.
		00	Full rate. Two data samples taken per PLL output clock cycle.
		01	Half rate. One data sample taken per PLL output clock cycle.
		10	Quarter rate. One data sample taken every two PLL output clock cycles.
		11	Reserved

Table 10. Bits of SERDES_CFGTX*n*_CNTL Registers (continued)

Bit	Field	Value	Description
4:2	BUS-WIDTH		Bus width. Selects the width of the parallel interface (10 or 8 bit).
		000	10-bit operation. Data is input on TD*n*[9:0]. TXBCLK*n* period is 10 bit periods (4 high, 6 low).
		001	8-bit operation. Data is input on TD*n*[9:2]. TXBCLK*n* period is 8 bit periods (4 high, 4 low). TD*n*[1:0] are ignored.
		01x	Reserved
		1xx	Reserved
1	Reserved		Reserved
0	ENTX		Enable transmitter. Enables this transmitter when high.

Table 11. SWING Bits

CFGTX[11:9]	Amplitude (mV_{dfpp})
000	125
001	250
010	500
011	625
100	750
101	1000
110	1125
111	1250

Table 12. DE Bits

CFGTX[15:12]	Amplitude Reduction	
	%	dB
0000	0	0
0001	4.76	-0.42
0010	9.52	-0.87
0011	14.28	-1.34
0100	19.04	-1.83
0101	23.8	-2.36
0110	28.56	-2.92
0111	33.32	-3.52
1000	38.08	-4.16
1001	42.85	-4.86
1010	47.61	-5.61
1011	52.38	-6.44
1100	57.14	-7.35
1101	61.9	-8.38
1110	66.66	-9.54
1111	71.42	-10.87

2.3.2.4 SERDES Configuration Example

```
rdata = SRIO_REGS->SERDES_CFG0_CNTL;
wdata = 0x00000001;
mask  = 0x00000FFF;
mdata = (wdata & mask) | (rdata & ~mask);
SRIO_REGS->SERDES_CFG0_CNTL     = mdata ;  // 3.125 Gbps
SRIO_REGS->SERDES_CFG1_CNTL     = mdata ;  // 3.125 Gbps
SRIO_REGS->SERDES_CFG2_CNTL     = mdata ;  // 3.125 Gbps
SRIO_REGS->SERDES_CFG3_CNTL     = mdata ;  // 3.125 Gbps

SRIO_REGS->SERDES_CFGRX0_CNTL   = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX1_CNTL   = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX2_CNTL   = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX3_CNTL   = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGTX0_CNTL   = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX1_CNTL   = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX2_CNTL   = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX3_CNTL   = 0x00010801 ;  // enable tx, rate 1
```

2.3.3 DirectIO

The DirectIO (Load/Store) module serves as the source of all outgoing direct I/O packets. With Direct I/O, the RapidIO packet contains the specific address where the data should be stored or read in the destination device. Direct I/O requires that a RapidIO source device, must keep a local table of addresses for memory within the destination device. If a CR ASIC is talking with DSP, the ASIC will have destination circular buffer description tables that contain DSP addresses, buffer sizes, and write pointer information. These tables are initialized by the DSP upon system boot after the initialization/discovery phase. Updates to the table could be managed completely by the DSP through RapidIO master writes. Once these tables are established, the ASIC RapidIO controller uses this data to compute the destination address and insert it into the packet header. The DSP RapidIO peripheral extracts the destination address from the packet header and transfers the payload to L2 memory via the DMA.

When a CPU wants to send data from memory to an external processing element (PE) or read data from an external PE, it must provide the RIO peripheral vital information about the transfer such as DSP memory address, target deviceID, target destination address, packet priority, etc. Essentially, a means must exist to fill all the header fields of the RapidIO packet. The Load/Store module provides a mechanism to handle this information exchange via a set of MMRs acting as transfer descriptors. These registers are addressable by the CPU through the configuration bus. Upon completion of a write to LSU_Reg5, a data transfer is initiated for either an NREAD, NWRITE, NWRITE_R, SWRITE, ATOMIC, or MAINTENANCE RapidIO transaction. Some fields, such as the RapidIO *srcTID/targetTID* field, are assigned by hardware and do not have a corresponding command register field.

Figure 9. Load/Store Data Transfer Diagram

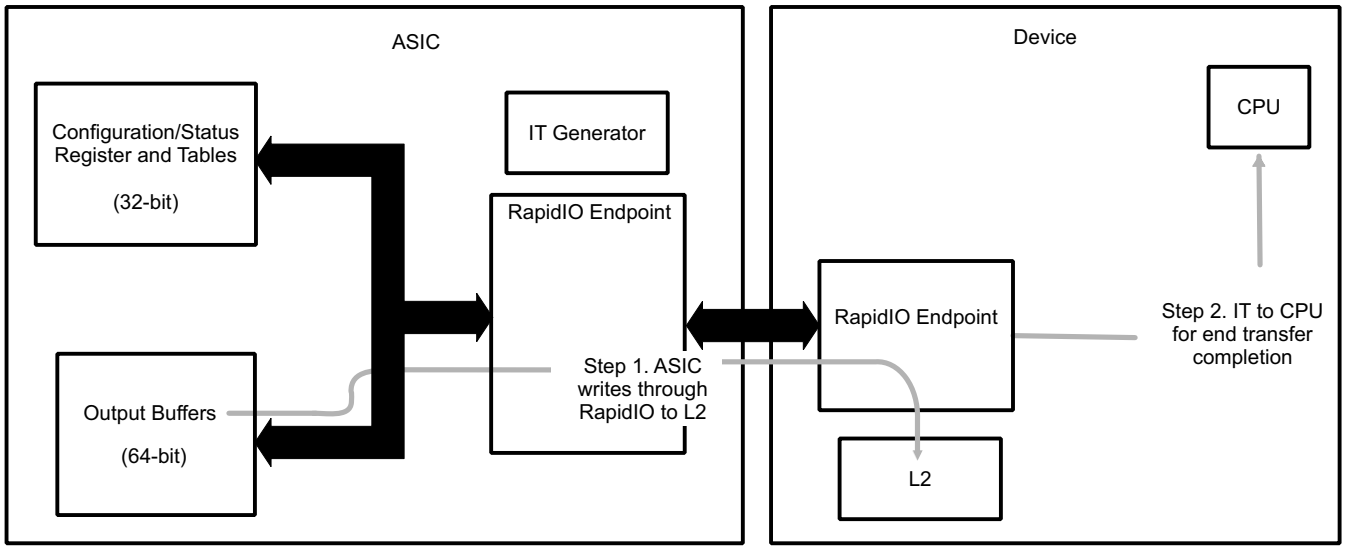
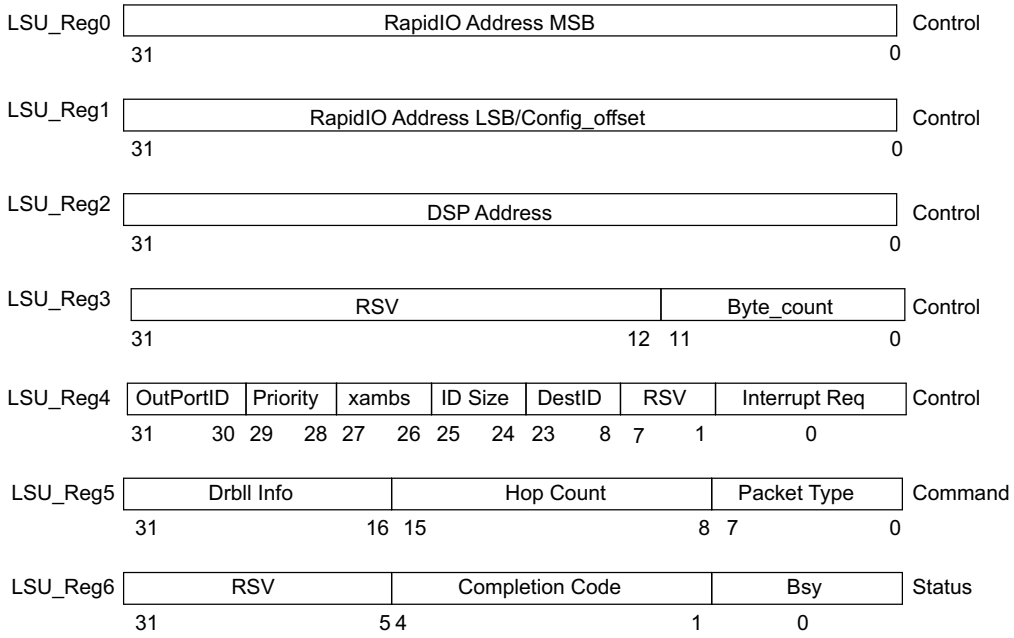


Figure 10. Load/Store Registers for RapidIO (Address Offset: LSU1 0x400-0x418, LSU2 0x420-0x438, LSU3 0x440-0x458, LSU4 0x460-0x478)



Mapping of command register fields to RapidIO packet header fields is as follows:

Table 13. Control/Command Register Field Mapping

Control/Command Register Field	RapidIO Packet Header Field
RapidIO Address MSB	32b Ext Address Fields – Packet Types 2,5, and 6
RapidIO Address LSB/Config_offset	1. 32b Address– Packet Types 2,5, and 6 (Will be used in conjunction with BYTE_COUNT to create 64b aligned RapidIO packet header address) 2. 24b Config_offset Field – Maintenance Packets Type 8 (Will be used in conjunction with BYTE_COUNT to create 64b aligned RapidIO packet header Config_offset). The 2 LSB of this field must be zero since the smallest configuration access is 4B.
DSP Address	32b DSP byte address. Not available in RapidIO Header.
Byte_Count	Number of data bytes to Read/Write - up to 4KB. (Used in conjunction with RapidIO address to create WRSIZE/RDSIZE and WDPTR in RapidIO packet header.) 000000000000b – 4KB 000000000001b – 1B 000000000010b – 2B . . . 111111111111b – 4095B (Maintenance requests are limited to 4B)
ID Size	RapidIO tt field specifying 8- or 16-bit DeviceIDs. 00b – 8b deviceIDs 01b – 16b deviceIDs 10b - reserved 11b - reserved
Priority	RapidIO prio field specifying packet priority (0 = lowest, 3 = highest). Request packets should not be sent at a priority level of 3 to avoid system deadlock. It is the responsibility of the software to assign the appropriate outgoing priority.
Xambs	RapidIO xambs field specifying extended address MSB.
DestID	RapidIO destinationID field specifying target device.

Table 13. Control/Command Register Field Mapping (continued)

Control/Command Register Field	RapidIO Packet Header Field
Packet Type	4 msb = 4b ftype field for all packets and 4 lsb = 4b trans field for packet types 2,5,8.
OutPortID	Not available in RapidIO header. Indicates the output port number for the packet to be transmitted from. Specified by the CPU along with NodeID.
Drbll Info	RapidIO doorbell info field for type 10 packets.
Hop Count	RapidIO hop_count field specified for Type 8 Maintenance packets.
Interrupt Req	Not available in RapidIO header. CPU controlled request bit used for interrupt generation. Typically used in conjunction with non-posted commands to alert the CPU when the requested data/status is present. 0b - An interrupt is not requested upon completion of command 1b- An interrupt is requested upon completion of command

Table 14. Status Fields

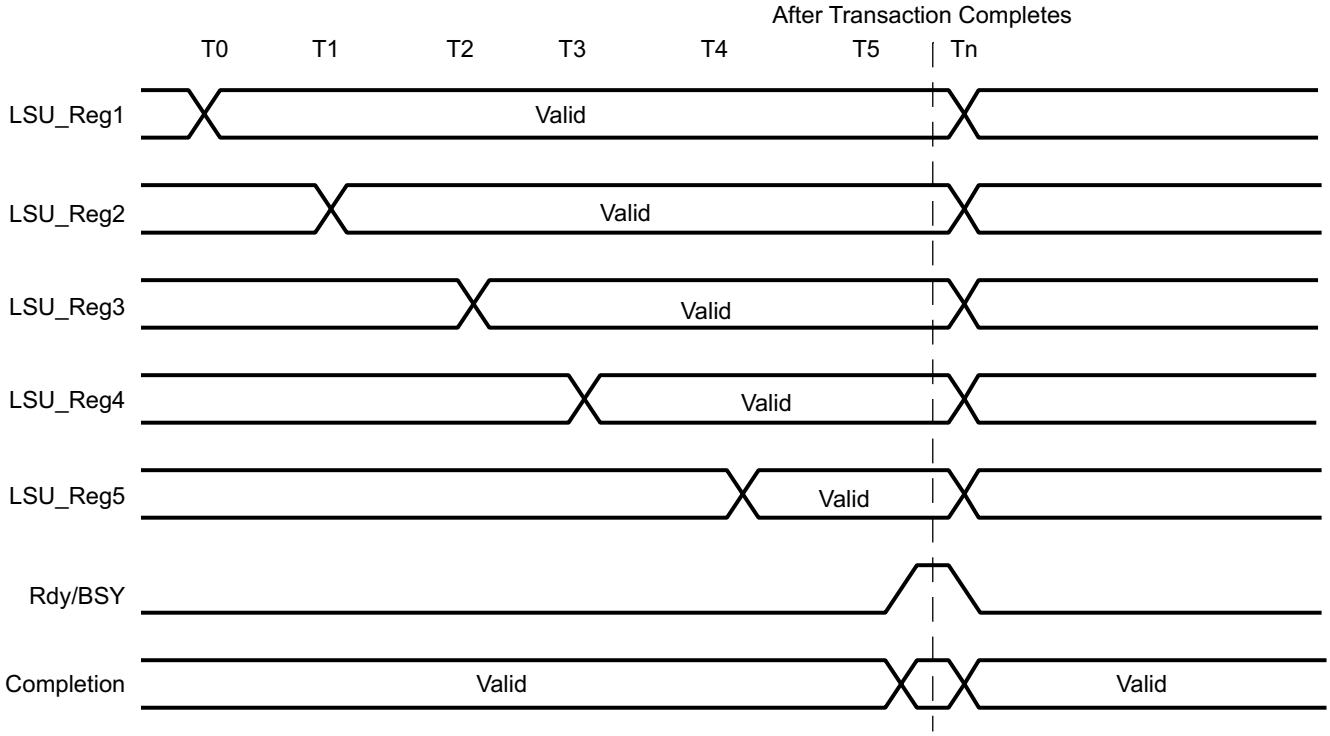
Status Field	Function
BSY	Indicates status of the command registers. 0b - Command registers are available (writable) for next set of transfer descriptors 1b - Command registers are busy with current transfer
Completion Code	Indicates the status of the pending command. 000b – Transaction complete, no errors (Posted/Non-posted) 001b – Transaction timeout occurred on Non-posted transaction 010b – Transaction complete, packet not sent due to flow control blockade (Xoff) 011b – Transaction complete, non-posted response packet (type 8 and 13) contained ERROR status, or response payload length was in error 100b – Transaction complete, packet not sent due to unsupported transaction type or invalid programming encoding for one or more LSU register fields 101b – DMA data transfer error 110b – Retry DOORBELL response received, or Atomic Test-and-swap was not allowed (semaphore in use) 111b – Transaction complete, packet not sent due to unavailable outbound credit at given priority (1)

(1) Status available only when Bsy signal = 0.

Four LSU register sets exist. This allows four outstanding requests for all transaction types that require a response (i.e., non-posted). For multi-core devices, software manages the usage of the registers. A shared configuration bus accesses all register sets. A single core device can utilize all four LSU blocks.

Figure 11 shows the timing diagram for accessing the LSU registers. Bsy signal is deasserted. LSU_Reg1 is written on configuration bus clock cycle T0, LSU_Reg2 is written on cycle T1, LSU_Reg3 is written on cycle T2, LSU_Reg4 is written on cycle T3. The command register LSU_Reg5 is written on cycle T4. The extended address field in LSU_Reg0 is assumed to be constant in this example. Upon completion of the write to the command register (next clock cycle T5), the Bsy signal is asserted, at which point the preceding completion code is invalid and accesses to the LSU registers are not allowed. Once the transaction completes (either as a successful transmission, or unsuccessfully, such as flow control prevention or response timeout) and any required interrupt service routine is completed, the Bsy signal is deasserted and the completion code becomes valid and the registers are accessible again.

Figure 11. LSU Registers Timing

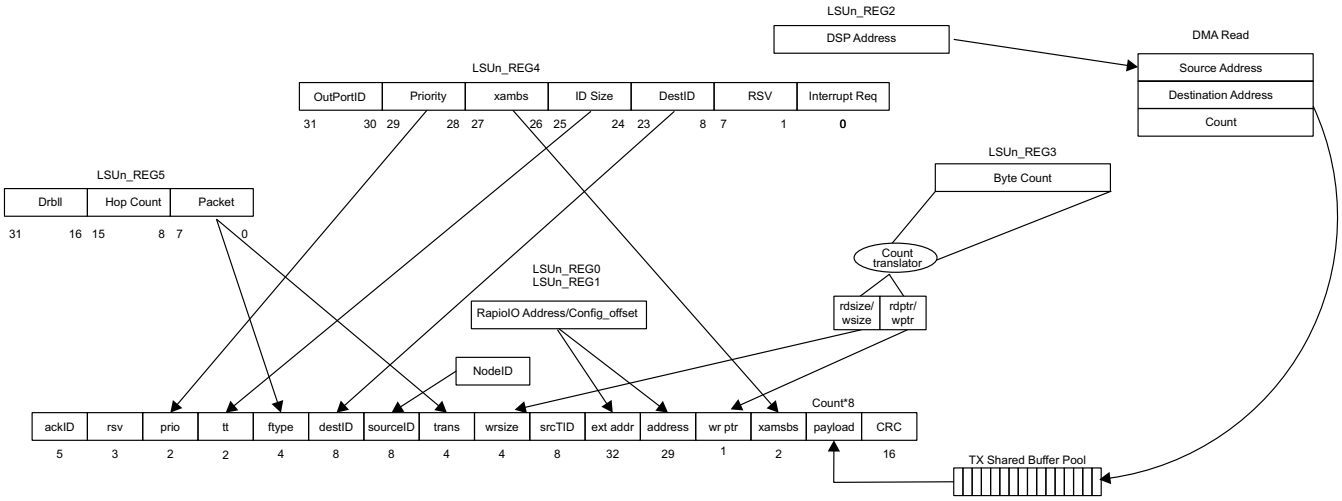


The following code illustrates an LSU registers programming example.

```
SRIO_REGS->;LSU1_Reg0 =     CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
           //poll mode, extended address type 2,5,6
SRIO_REGS->;LSU1_Reg1 =     CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET,(int);rcvBuff1[0] );
           //32bit = type 2,5,6. 24bit = type 8
SRIO_REGS->;LSU1_Reg2 =     CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, (int);xmtBuff1[0]);
SRIO_REGS->;LSU1_Reg3 =     CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT,byte_count );
SRIO_REGS->;LSU1_Reg4 =     CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,0 )|
                            CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )|
                            CSL_FMK( SRIO_LSU1_REG4_XAMBS,0 )|
                                 //no extended address
                            CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )|
                                 //tt = 0b01
                            CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )|
                            CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,1 );
                                 //0 = event-driven, 1 = poll
SRIO_REGS->;LSU1_Reg5 =     CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                            CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x00 )|
                            CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );
                                //REQ_NWRITE
```

Figure 12 illustrates an example of the data flow and field mappings for a burst NWRITE_R transaction:

Figure 12. Example Burst NWRITE_R



For WRITE commands, the payload is combined with the header information from the control/command registers and buffered in the shared TX buffer resource pool. Finally, it is forwarded to the TX FIFO for transmission. READ commands have no payload. In this case, only the control/command register fields are buffered and used to create a RapidIO NREAD packet, which is forwarded to the TX FIFO. Corresponding response packet payloads from READ transactions are buffered in the shared RX buffer resource pool when forwarded from the receive ports. Both posted and non-posted operations rely on the OutPortID command register field to specify the appropriate output port/FIFO.

The data is burst internally to the Load/Store module at the DMA clock rate.

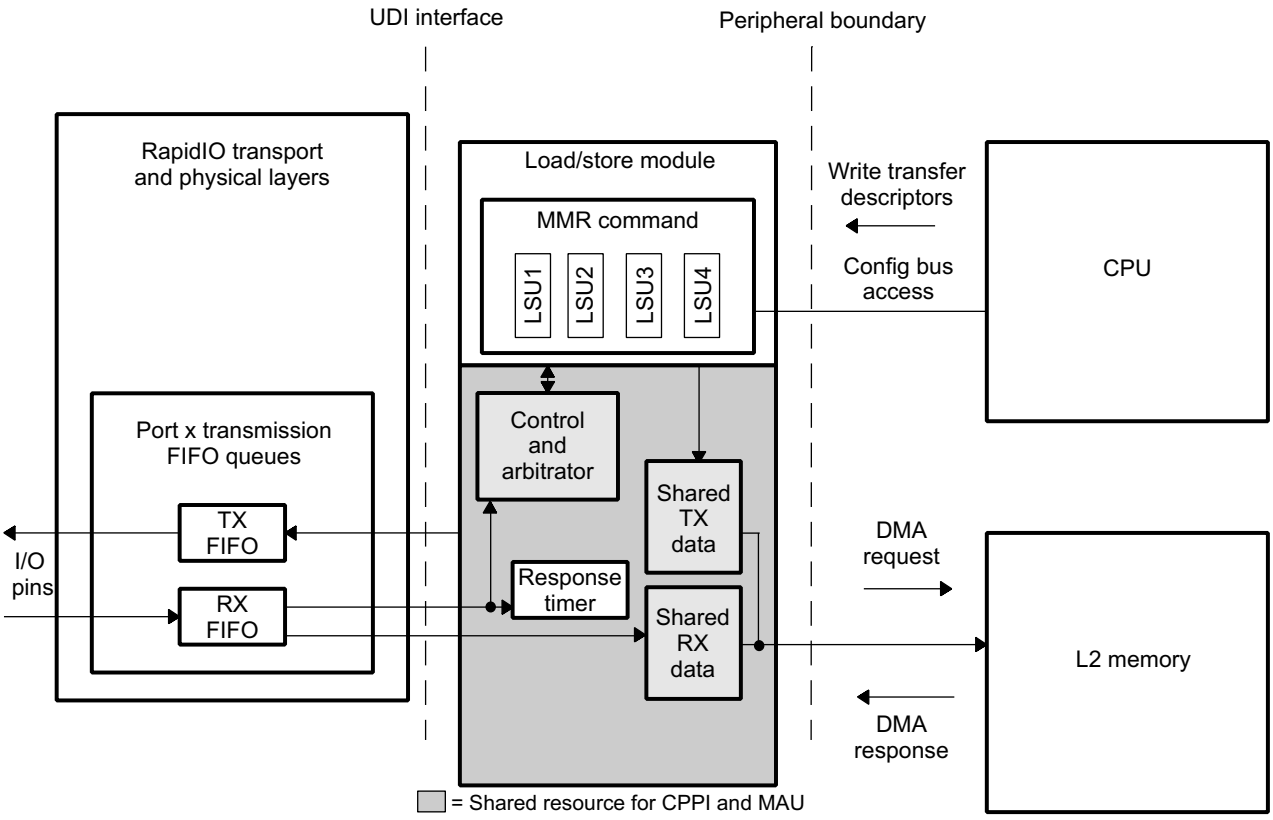
2.3.3.1 Detailed Data Path Description

The Load/Store module is for generating all outgoing RapidIO Direct I/O packets. Any read or write transaction, other than the messaging protocol, uses this interface. In addition, outgoing DOORBELL packets are generated through this interface.

The data path for this module uses DMA bus as the DMA interface. The configuration bus is used by the CPU to access the control/command registers. The registers contain transfer descriptors that are needed to initiate READ and WRITE packet generation. After the transfer descriptors are written, flow control status is queried. The unit examines the DESTID and PRIORITY fields of LSU*n*_REG4 to determine if that flow has been Xoffd. Additionally, the free buffer status of the TX FIFO is checked (based on the OutPortID register field). Only after the flow control access is granted, and a TX FIFO buffer has been allocated, can a DMA bus read command be issued for payload data to be moved into the shared TX buffer pool. Data is moved from the shared buffer pool to the appropriate output TX FIFO in simple sequential order based on completion of the DMA bus transaction. However, if fabric congestion occurs, priority can affect the order in which the data leaves the TX FIFOs.

Here a reordering mechanism exists, which transmits the highest priority packets first if RETRY acknowledges. Once in the FIFO, the data is guaranteed to be transmitted through the pins. Alternatively, if an intended flow has been shut down, the peripheral signals the CPU with an interrupt to notify that the packet was not sent and sets the completion code to 010b in the status register. The registers must be held until the interrupt service routine is complete before the BSY signal is released (BSY=0 in LSU*n*_REG6) and the CPU can then rewrite or overwrite the transfer descriptors with new data. Figure 13 illustrates the data path and buffering that is required to support the Load/Store Module.

Figure 13. Load/Store Module Data Flow



2.3.3.2 TX Operation

WRITE Transactions:

The TX buffers are implemented in a single SRAM and shared between multiple cores. A state machine arbitrates and assigns available buffers between the LSUs. When the DMA bus read request is transmitted, the appropriate TX buffer address is specified within it. The data payload is written to that buffer through the DMA bus response transaction. Depending on the architecture of the device, interleaving of multi-segmented DMA bus responses from the DMA is possible. Upon receipt of a DMA bus read response segment, the unit checks the completion status of the payload. Note that only one payload can be completed in any single DMA bus cycle. The Load/Store module can only forward the packet to the TX FIFO after the final payload byte from the DMA bus response has been written into the shared memory buffer. Once the packet is forwarded to the TX FIFO, the shared buffer can be released and made available for a new transaction.

The TX buffer space is dynamically shared among all outgoing sources, including the Load/Store Unit (LSU) and the TX CPPI, as well as the response packets from RX CPPI and the Memory Access Unit (MAU). Thus, the buffer space memory must be partitioned to handle packets with and without payloads. A 4.5KB buffer space is configured to support 16 packets with payloads up to 256B, in addition to 16 packets without payloads. The SRAM is configured as a 128-bit wide two port, which matches the UDI width of the TX FIFOs.

Data leaves the shared buffer pool sequentially in order of receipt, not based on the packet priority. However, if fabric congestion occurs, priority can affect the order in which the data leaves the TX FIFOs. A reordering mechanism exists here, which transmits the highest priority packets first if RETRY acknowledges.

For posted WRITE operations, which do not require a RapidIO response packet, a core may submit multiple outstanding requests. For instance, a single core may have many streaming write packets buffered at any given time, given outgoing resources. In this application, the control/command registers can be released (BSY = 0) to the CPU as soon as the header info is written into the shared TX buffer pool. If the request has been flow controlled, the peripheral will set the completion code status register and appropriate interrupt bit of the ICSR. The control/command registers can be released after the interrupt service routine completes.

For non-posted WRITE operations, which do require a RapidIO response packet, there can be only one outstanding request per core at any given time. The payload data and header information is written to the shared TX buffer pool as described above; however, the command registers cannot be released (BSY = 1) until the response packet is routed back to the module and the appropriate completion code is set in the status register. One special case exists for outgoing test-and-swap packets (Ftype 5, Transaction 1110b). This is the only WRITE class packet that expects a response with payload. This response payload is routed to the LSU, it is examined to verify whether the semaphore was accepted, and then the appropriate completion code is set. The payload is not transferred out of the peripheral via the DMA bus.

So the general flow is as follows:

- LSU registers are written using the configuration bus
- Flow control is determined
- TX FIFO free buffer availability is determined
- DMA bus read request for data payload
- DMA bus response writes data to specified module buffer in the shared TX buffer space
- DMA bus read response is monitored for last byte of payload
- Header data in the LSU registers is written to the shared TX buffer space
- Payload and header are transferred to the TX FIFO
- The LSU registers are released if no RapidIO response is needed
- Transfer from the TX FIFO to external device based on priority

READ Transactions:

The flow for generating READ transactions is similar to non-posted WRITE with response transactions. There are two main differences: READ packets contain no data payload, and READ responses have a payload. So READ commands simply require a non-payload TX buffer within the shared pool. In addition, they require a shared RX data buffer. This buffer is not pre-allocated before transmitting the READ request packet, since doing so could cause traffic congestion of other in-bound packets destined to other functional blocks.

Again, the control/command registers cannot be released (BSY = 1) until the response packet is routed back to the module and appropriate completion code is set in the status register.

So the general flow would be:

- LSU registers are written using the configuration bus
- Flow control is determined
- TX FIFO buffer is allocated
- Header data in the LSU registers is written to the shared TX buffer
- Payload and header are transferred to the TX FIFO
- The LSU registers are released if no RapidIO response is needed
- Transfer from the TX FIFO to external device based on priority

For all transactions, the shared TX buffers are released as soon as the packet is forwarded to the TX FIFOs. If an ERROR or RETRY response is received for a non-posted transaction, the CPU must either reinitiate the process by writing to the LSU register, or initiate a new transaction altogether.

Segmentation:

The LSU handles two types of segmentation of outbound requests. The first type is when the Byte_Count of Read/Write requests exceeds 256 bytes (up to 4KB). The second type is when Read/Write request RapidIO address is non-64b aligned. In both cases, the outgoing request must be broken up into multiple RapidIO request packets. For example, assume that the CPU wants to perform a 1KB store operation to an external RapidIO device. After setting up the LSU registers, the CPU performs one write to the LSU_Reg5 register. The peripheral hardware then segments the store operation into four RapidIO write packets of 256B each, and calculates the 64b-aligned RapidIO address, WRSIZE, and WDPTR as required for each packet. This example requires four outbound handles to be assigned and four DMA transmit requests. The LSU registers cannot be released until all posted request packets are passed to the TX FIFOs. Alternatively, for non-posted operations, such as CPU loads, all packet responses must be received before the LSU registers are released.

2.3.3.3 RX Operation

Response packets are always type 13 RapidIO packets. All response packets with transaction types not equal to 0001b are routed to the LSU block sequentially in order of reception. These packets may have a payload, depending on the type of corresponding request packet that was originally sent. Due to the nature of RapidIO switch fabric systems, response packets can arrive in any order. The data payload, if any, and header data is moved from the RX FIFO to the shared RX buffer. The DestID field of the packet is examined to determine which core and corresponding set of registers are waiting for the response. Remember, there can be only one outstanding request per core. Any payload data is moved from the shared RX buffer pool into memory through normal DMA bus operations.

Registers for all non-posted operations should only be held for a finite amount of time to avoid blocking resources when a request or response packet is somehow lost in the switch fabric. This time correlates to the 24-bit Port Response Time-out Control CSR value discussed in sections 5.10.1 and 6.1.2.4 of the serial specification. If the time expires, control/command register resources should be released, and an error is logged in the ERROR MANAGEMENT RapidIO registers. The RapidIO specification states that the maximum time interval (all 1s) is between 3 and 6 seconds. A logical layer timeout occurs if the response packet is not received before a countdown timer (initialized to this CSR value) reaches zero.

Each outstanding packet response timer requires a 4-bit register. The register is loaded with the current timecode when the transaction is sent. Each time the timecode changes, a 4-bit compare is done to the register. If the register becomes equal to the timecode again, without a response being seen, then the transaction has timed out. Essentially, instead of the 24-bit value representing the period of the response timer, the period is now defined as P = (2^24 x 16)/F. This means the countdown timer frequency needs to be 44.7 – 89.5Mhz for a 6 – 3 second response timeout. Because the needed timer frequency is derived from the DMA bus clock (which is device dependent), the hardware supports a programmable configuration register field to properly scale the clock frequency. This configuration register field is described in the Peripheral Setting Control register (Address offset 0x0020).

If a response packet indicates ERROR status, the Load/Store module notifies the CPU by generating an error interrupt for the pending non-posted transaction. If the response has completed successfully, and the Interrupt Req bit is set in the control register, the module generates a CPU servicing interrupt to notify the CPU that the response is available. The control/command registers can be released as soon as the response packet is received by the logical layer. The hardware is not responsible for attempting a retransmission of the non-posted transaction.

If a Doorbell response packet indicates Retry status, the Load/Store module notifies the CPU by generating an interrupt. The control/command registers can be released as soon as the response packet is received by the logical layer. The hardware is not responsible for attempting retransmission of the Doorbell transaction.

So the general flow is as follows:

- Previously, the control/command registers were written and the request packet was sent
- Response Packet Type13, Trans != 0001b arrives at module interface, and is handled sequentially (not based on priority)
- targetTID is examined to determine routing of a response to the appropriate core
- The status field of the response packet is checked for ERROR, RETRY or DONE
- If the field is DONE, it submits DMA bus request and transmits the payload (if any) to DSP address. If the field is ERROR/RETRY, it sets an interrupt
- Command registers are released (BSY = 0)
- Optional Interrupt to CPU notifying packet reception

2.3.3.4 Reset and Power Down State

Upon reset, the Load/Store module must clear the command register fields and wait for a write by the CPU.

The Load/Store module can be powered down if the direct I/O protocol is not being supported in the application. For example, if the messaging protocol is being used for data transfers, powering down the Load/Store module will save power. In this situation, the command registers should be powered down and inaccessible. Clocks should be gated to these blocks while in the power down state.

2.3.4 Message Passing

With message passing, a destination address is not specified. Instead, a mailbox identifier is used within the RapidIO packet. The mailbox is controlled and mapped to memory by the local (destination) device. The message passing within RapidIO specifies 4 mailbox locations. Each mailbox can contain 4 separate transactions (or letters), effectively providing 16 messages. Single packet messages provide 64 mailboxes with 4 letters, effectively providing 256 messages. Mailboxes can be defined for different data types or priorities. The advantage of message passing is that the source device does not require any knowledge of the destination device's memory map. The DSP contains Buffer Descriptions Tables for each mailbox. These tables define a memory map and pointers for each mailbox. Messages are transferred to the appropriate memory locations via the DMA.

The CPPI (Communications Port Programming Interface) module serves as the incoming and outgoing message passing protocol engine.

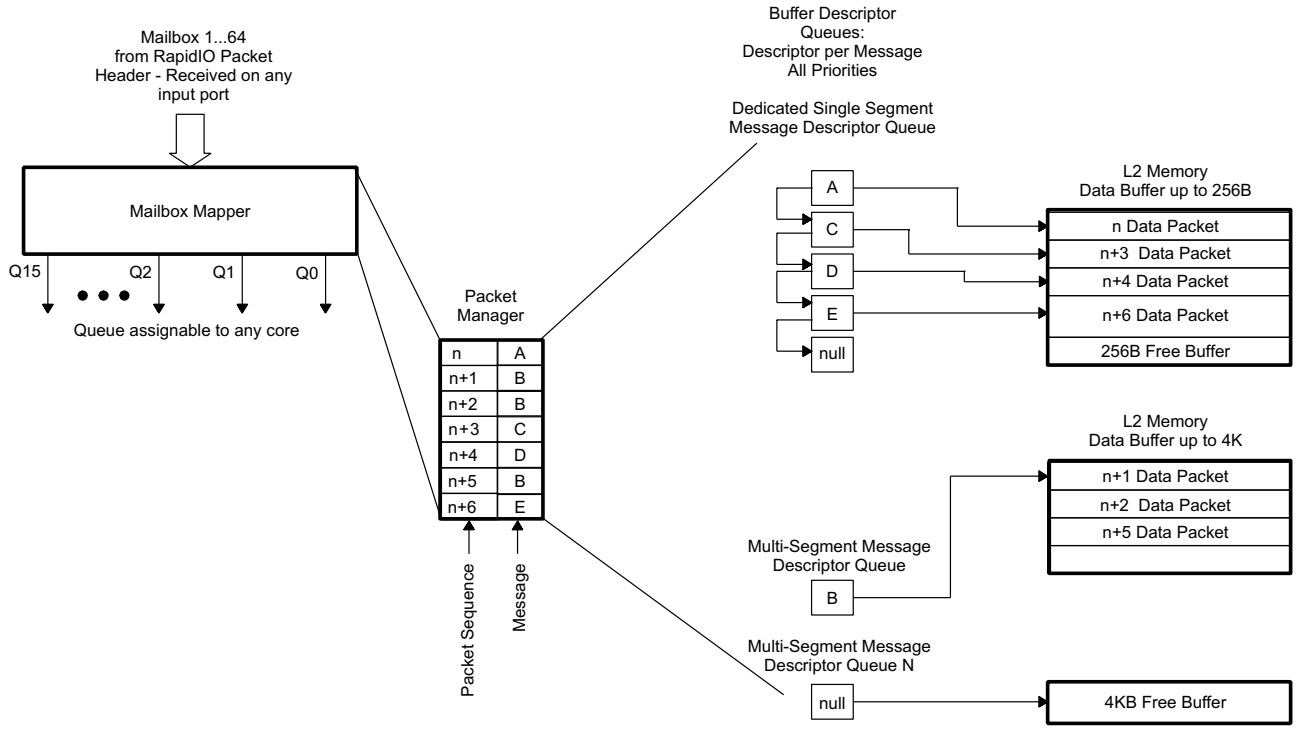
The following rules exist for all CPPI traffic:

- One buffer descriptor per message (each buffer descriptor consists of 4 words or 16 bytes)
- Requires contiguous memory space for multi-segment Read/Write operations
  - Fixed buffer sizes (configured to handle application's max message size)
- An ERROR response is sent if the RX message is too big for the allotted buffer sizes
  - Subsequent ERROR responses will be sent for all segments of that message
- An ERROR response is sent if the mailbox is not mapped, or mapped to a non-existent queue
- An ERROR response is sent if the mailbox is mapped, but the queue is not initialized (RX DMA State HDP not written), or the queue is disabled (Teardown)
- An ERROR response is sent if the RX buffer descriptor queue has no empty buffers (overflow)
- Out-of-order responses are allowed
- RETRY response will be issued to the first received segment of a multi-segment message when the RX queue is busy servicing another request
  - Subsequent RETRY responses may have to be sent for received pipeline segments or additional pipelined messages to the same queue
- Inorder message reception for dedicated flows is mode programmable
- A queue is needed for each supported simultaneous multi-segment RX message
- Supports a minimum of 1.25KB SRAM (64 buffer descriptors)
- Transmit must be able to RETRY any given segment of a transmitted message
- A Dest_id is equal to port for TX operations, the same Dest_Id is not accessible from multiple ports

2.3.4.1 RX Operation

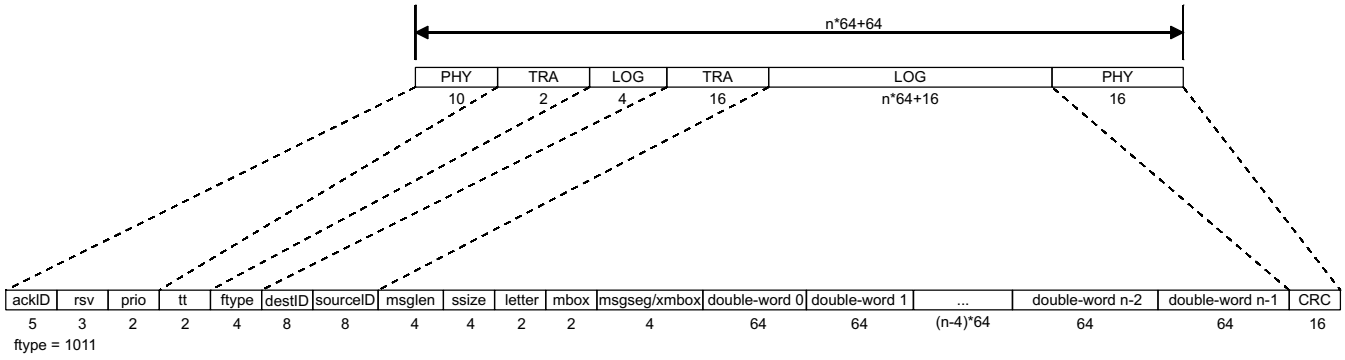
As message packets are received by the RapidIO ports, the data must be written into memory while maintaining accurate state information that is needed for future processing. For instance, if a message spans multiple packets, information must be saved that allows re-assembly of those packets by the CPU. The CPPI module provides a scheme for tracking single and multi-packet messages, linking messages in queues, and generating interrupts. Figure 14 illustrates the scheme.

Figure 14. CPPI RX Scheme for RapidIO



Messages addressed to any of the 64 mailbox locations can be received on any of the RapidIO ports simultaneously. Packets are handled sequentially in order of receipt. The function of the mailbox mapper block is to direct the inbound messages to the appropriate queue and finally to the correct core. The queue mapping is programmable and must be configured after device reset. RapidIO originally supported only 4 mailboxes with 4 letters/mailbox. Letters allow concurrent message traffic between sender and receiver. However, for messages that consist of only single packets, the unused 4-bit packet field normally indicating the message segment extends the available number of mailboxes. Figure 15 shows the packet header fields for message requests.

Figure 15. Message Request Packet



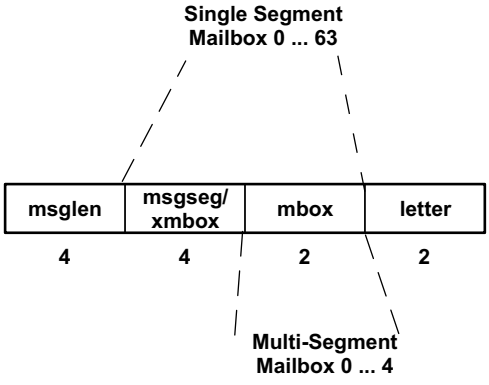
This allows the letter and mailbox fields to instead allow four concurrent single-segment messages to sixty-four possible mailboxes (256 total locations) for a source and destination pair. The mailbox mapper directs the inbound messages to the appropriate queue based on a pre-programmed routing table. It bases the decision on the SOURCEID, MSGLEN, MBOX, LETTER, and XMBOX fields of the RapidIO packet.

Figure 16 illustrates the look-up tables required for programmable mapping of the mailbox to queue. There are 32 programmable mapping entries. Each mapping entry consists of two registers, RXU_MAP_L*n* and RXU_MAP_H*n*. Each entry stores the queue number associated with the message's intended mailbox/letter. If a mailbox/letter is not supported or does not have a mapping table entry, the message is discarded and an ERROR response sent. The mapping entries can explicitly call out a mailbox and letter combination, or alternatively, the mask fields can be used to grant multiple mailbox/letter combinations access to a queue using the same table entry. A masking value of 0 in the mailbox or letter mask fields indicates that the corresponding bit in the mailbox or letter field will not be used to match for this queue mapping entry. For example, a mailbox mask of all zeros would allow a mapping entry to be used for all incoming mailboxes.

The mapping table entry also provides a security feature to enable or disable access from specific external devices to local mailboxes. The SOURCEID field indicates which external device has access to the mapping entry and corresponding queue. A compare is performed between the sourceID of the incoming message packet and each relevant mailbox/letter table mapping entry SOURCEID field. If they do not match, an ERROR response is sent back to the sender, and the transaction is logged in the Logical Layer Error Management capture registers, which sets an interrupt, as discussed in Section 4.3. A Promiscuous bit allows this security feature to be disabled. When the PROMISCUOUS bit is set, full access to the mapping entry from any SOURCEID is allowed. Note that when the PROMISCUOUS bit is set, the mailbox/letter and corresponding mask bits are still in effect. When the PROMISCUOUS bit is cleared, it equals a mask value of 0xFFFF, and only the matching SOURCEID is allowed access to the mailbox.

Each table entry also indicates if it used for single or multi-segment message mapping. Single segment message mapping entries utilize all six bits of the mailbox and corresponding mask fields. Multi-segment uses only the 2 LSBS. The number of simultaneous supported multi-segment messages is determined by the number of dedicated RX queues as discussed further below. It is recommended to dedicate a multi-segment mapping entry for each supported simultaneous letter. Essentially, letter masks should be avoided for multi-segment mapping to reduce excessive retries. Note that it is possible to configure the table entries such that incoming single segment and multi-segment messages are directed to the same queue. To avoid this condition, properly program the mapping table entries.

Figure 16. Queue Mapping Table (Address Offset: 0x0800 - 0x08FC)

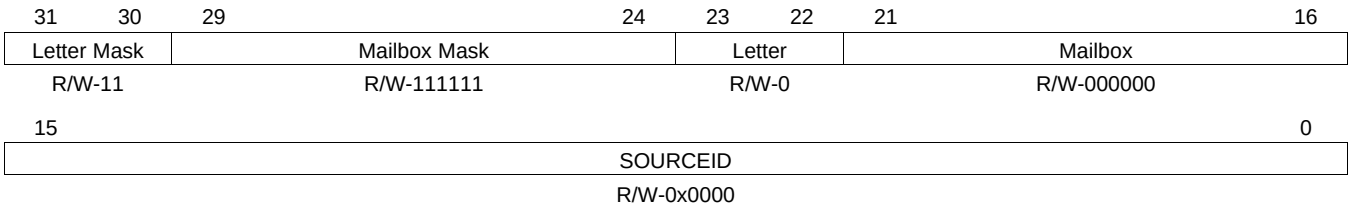


SOURCEID = SourceID allowed access if secure queue
Mailbox = Allowed mailbox for this mapping register
(Mask-able)
0b000000 - Mailbox 0
0b000001 - Mailbox 1
0b000010 - Mailbox 2
...
0b111111 - Mailbox 63

Letter = Allowed letter for this mapping register (Mask-able)

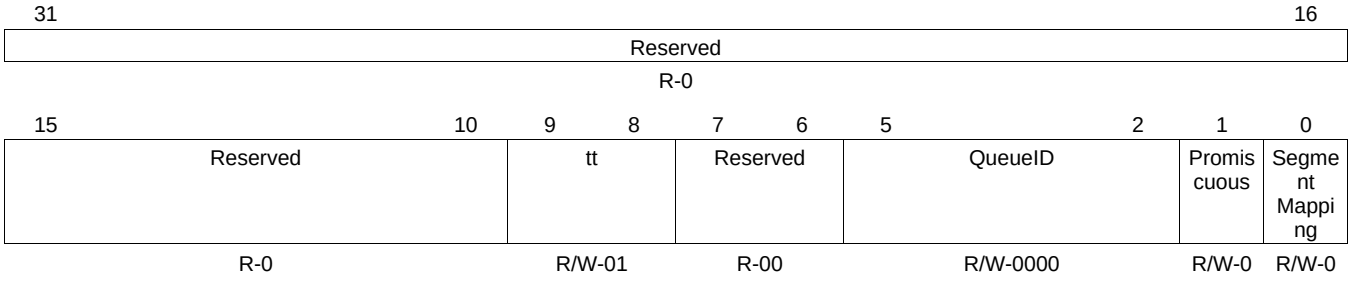
Segment Mapping = 0b0 - Single Segment (all six bits of Mailbox valid)
= 0b1 - Multi-segment (only 2 lsbs of Mailbox valid)
Promiscuous = 1, Full Access to the Queue for any SourceID
Queue ID = 0000 - 1111, corresponding Queue0 - Queue15
tt (Transport type) - 0b0 = matches the 8 lsb of the SOURCEID
0b1 = matches the full 16 bits of the SOURCEID

Figure 17. Queue Mapping Register RXU_MAP_L*n*



LEGEND: R = Read, W = Write, n = value at reset

Figure 18. Queue Mapping Register RXU_MAP_H*n*



LEGEND: R = Read, W = Write, n = value at reset

The packet manager maintains the RX DMA state of free and used data buffers within the memory space. It directs the data to specific addresses within the memory and maintains and updates the buffer descriptor queues. There is a single buffer descriptor per RapidIO message. For example, single segment messages have one buffer descriptor, as do multi-segment messages with up to 4KB payloads.

There can be multiple RX buffer descriptor queues per core. It is suggested that one queue be dedicated to single segment messages and additional queues be dedicated to multi-segment messages. Each multi-segment message queue can support only one incoming message at a time. Depending on the application, it may be necessary to support multiple simultaneous SAR operations per core. In this case, a buffer descriptor queue is allocated for each desired simultaneous message. The peripheral supports a total of 16 assignable RX queues and their associated RX DMA state registers. Each of the queues can be assigned to single or multi-segment messages.

Table 15 and Table 16 show the RX DMA State Registers.

Table 15. RX DMA State Head Descriptor Pointer (HDP) (Address Offset 0x600-0x63C)

Bit	Name	Description
31:0	RX Queue Head Descriptor Pointer	Rx Queue Head Descriptor Pointer: This field is the host memory address for the first buffer descriptor in the channel receive queue. This field is written by the host to initiate queue receive operations and is zeroed by the port when all free buffers have been used. An error condition results if the host writes this field when the current field value is nonzero. The address must be 32-bit word aligned.

Table 16. RX DMA State Completion Pointer (CP) (Address Offset 0x600-0x63C)

Bit	Name	Description
31:0	RX Queue Completion Pointer	Rx Queue Completion Pointer: This field is the host memory address for the receive queue completion pointer. This register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The port uses the value written to determine if the interrupt should be deasserted.

If a multi-segment buffer descriptor queue is not currently free, and an Rx port receives another multi-segment message that is destined for that queue, the RX CPPI must send a RETRY RESPONSE packet (Type 13) to the sender, indicating that an internal buffering problem exists. If a multi-segment buffer descriptor queue is busy and there is another incoming multi-segment message with the same SOURCEID, MAILBOX, and LETTER, an ERROR response is sent. This usually indicates that a TX programming error has occurred, where duplicate segments or segments outside the MSGLEN were sent. Upon successful reception of any message segment, the RX CPPI is responsible for sending a DONE response to the sender.

If a RX message's length is greater than that of the targeted buffer descriptor, an ERROR response is sent back to the source device. In addition, the DSP is notified with the use of the CC field of the RX CPPI buffer descriptor, described as follows. This situation can result from a DSP software error (misallocating a buffer for the queue), or as a result of sender error (sending to a wrong mailbox).

An Rx transaction timeout is used by all multi-segment queues, in order to not hang receive mailbox resources in the event that a message segment is lost in the fabric. This response-to-request timer controls the time-out for sending a response packet and receiving the next request packet of a given multi-segment message. It has the same value and is analogous to the request-to-response timer discussed in the TX CPPI and LSU sections, which is defined by the 24-bit value in the port response time-out CSR. The RapidIO specification states that the maximum time interval (all 1s) is between 3 and 6 seconds. Each multi-segment receive timer requires a 4-bit register. The register is loaded with the current timecode when the response is sent. Each time the timecode changes, a 4-bit compare is done to the register. If the register becomes equal to the timecode again, without the next message segment being seen, then the transaction has timed out. If this happens, the RX buffer resources can be released.

The buffer descriptor points to the corresponding data buffer in memory and also points to the next buffer descriptor in the queue. As segments of a received message arrive, the msgseg field of each segment is monitored to detect the completion of the received message. Once a full message is received, the OWNERSHIP bit is cleared in the packet's buffer descriptor to give control to the host. At this point, a host interrupt is issued. This interface works with programmable interrupt rate control, as discussed in Figure 57. There is an ICSR bit for each supported queue, as shown in Figure 47. On interrupt, the CPU processes the RX buffer queue, detecting received packets by the status of the OWNERSHIP bit in each buffer descriptor. The host processes the RX queue until it reaches a buffer descriptor with a set OWNERSHIP bit, or set EOQ bit. Once processing is complete, the host updates the RX DMA State Completion Pointer, allowing the peripheral to reuse the buffer.

Figure 19 shows the RX buffer descriptor fields. A RX buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit boundary. Accesses to these registers are restricted to 32-bit boundaries.

Figure 19. RX Buffer Descriptor Fields

Word Offset	Bit Fields																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Next Descriptor Pointer																															
1	Buffer Pointer																															
2	SRC_ID																PRI		tt		RESERVED									Mailbox		
3	SOP	EOP	OWNERSHIP	EOQ	TEARDOWN	RESERVED															cc			Message Length								

Table 17. RX Buffer Descriptor Field Descriptions

Field	Description
next_descriptor_pointer	Next Descriptor Pointer: The 32-bit word aligned memory address of the next buffer descriptor in the RX queue. This references the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The host sets the next_descriptor_pointer.
buffer_pointer	Buffer Pointer: The byte aligned memory address of the buffer associated with the buffer descriptor. The host sets the buffer_pointer.
Sop = 1	Start of Message: Indicates that the descriptor buffer is the first buffer in the message. • This bit will always be set, as this device only supports one buffer per message.
Eop = 1	End of Message: Indicates that the descriptor buffer is the last buffer in the message. • This bit will always be set, as this device only supports one buffer per message.
ownership	Ownership: Indicates ownership of the message and is valid only on sop. This bit is set by the host and cleared by the port when the message has been transmitted. The host uses this bit to reclaim buffers. 0: The message is owned by the host 1: The message is owned by the port
eoq	End Of Queue: Set by the port to indicate that the RX queue empty condition exists. This bit is valid only on eop. The port determines the end of queue condition by a zero next_descriptor_pointer. 0: The RX queue has more buffers available for reception. 1: The Descriptor buffer is the last buffer in the last message in the queue.
Teardown_Complete	Teardown Complete: Set by the port to indicate that the host commanded teardown process is complete, and the channel buffers may be reclaimed by the host. 0: The port has not completed the teardown process. 1: The port has completed the commanded teardown process.
message_length	Message Length: Initially written by the CPU to specify the maximum number of double-words the buffer can receive. Updated by the peripheral (after receiving a message) to indicate the actual number of double-words in the entire message. Message payloads are limited to a maximum size of 512 double-words (4096bytes). 000000000b: 512 double words 000000001b: 1 double word 000000010b: 2 double words . . . 111111111b: 511 double words
Src_id	Source Node ID: Unique node identifier of the source of the message.
tt	RapidIO tt field specifying 8- or 16-bit DeviceIDs 00: 8b deviceIDs 01: 16b deviceIDs 10: reserved 11: reserved
pri	Message Priority: Specifies the SRIO priority at which the message was sent.
cc	Completion Code: 000: Good completion. Message received. 001: Error, RX message length greater than supported buffer descriptor message_length 010: Error, TimeOut on receiving one of the segments 011: DMA transfer error on one or more segments 100: Queue teardown completed, data invalid 101: 111 Reserved

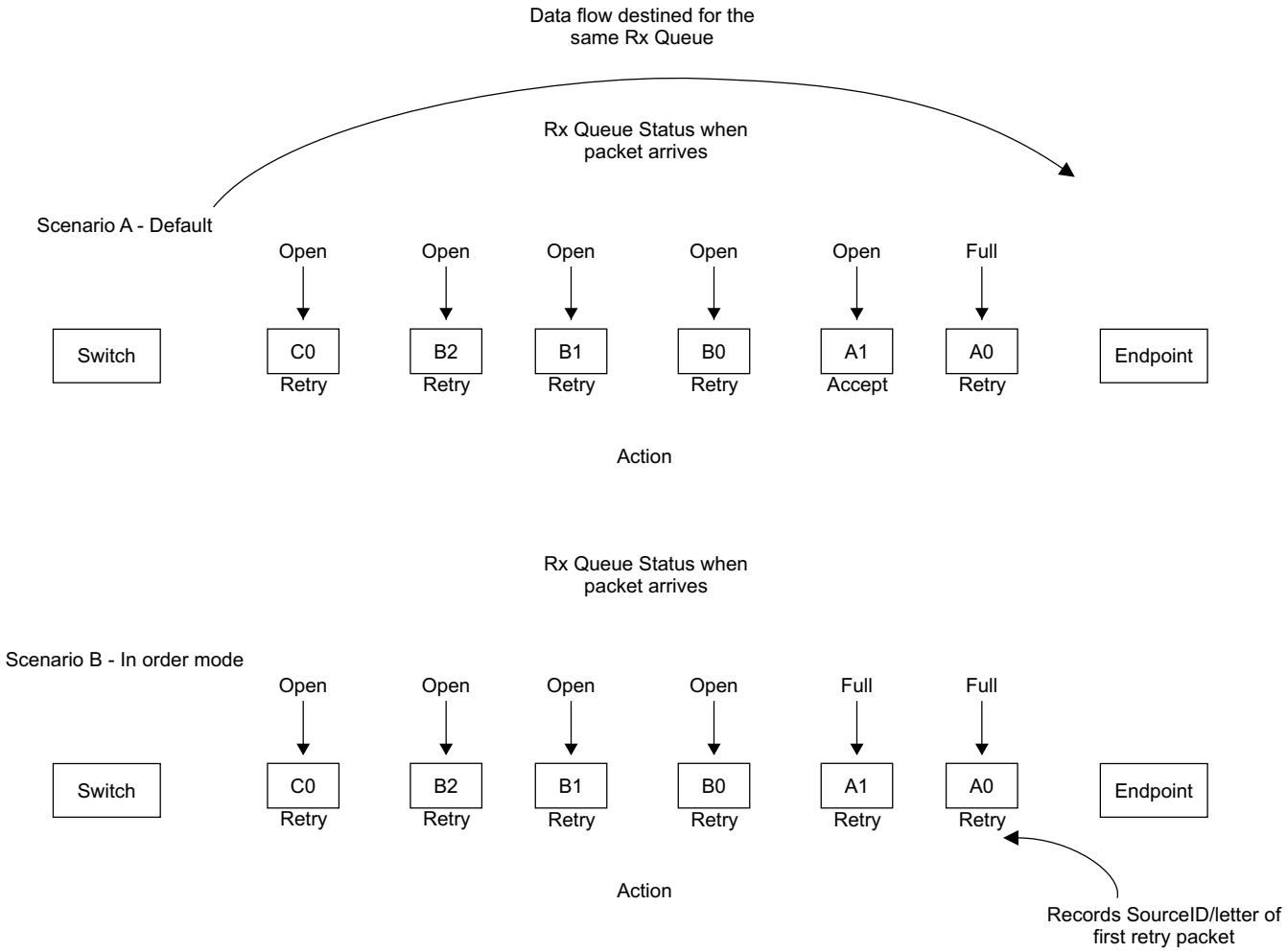
Table 17. RX Buffer Descriptor Field Descriptions (continued)

Field	Description
mailbox	Destination Mailbox: Specifies the mailbox to which the message was sent. 000000b: Mailbox 0 000001b: Mailbox 1 . . . 000100b: Mailbox 4 . . . 111111b: Mailbox 63 For multi-segment messages, only the two LSBs of this mailbox are valid. Hardware ignores the four MSBs if the incoming message has multiple segments.

Although the switch fabric must deliver the segments of multi-packet messages in the order they were sent, buffer resources at the receiving endpoint may only become available after the initial segment(s) of a message have had to be retried. The peripheral can accept out-of-order segments and track completion of the overall message. Scenario A in Figure 20 shows this concept.

For applications that are set up for specific message flows between a single source and destination, it may require in-order delivery of messages. This is described in scenario B of Figure 20. This scenario is similar to scenario A, although one message may be retried due to a lack of receiver resources, subsequent pipelined messages may arrive just as resources are freed up. This is a problem for systems requiring in-order message delivery. In this case, the peripheral needs to record the Src_id/mailbox/letter of the first retried message and retry all subsequent new requests until resources are available and a segment for that Src_id/mailbox/letter is received. As long as all messages are from the same source and that source sends (and retries) packets in order, then all messages will be received in order. Note that this solution is less effective when multiple sources share an RxQ. The RX CPPI Control register (Address offset 0x0744) sets this mode of operation on all receive queues. Once this mode is set and a retry is issued, the queue will continue to wait for an incoming message that matches the Src_id/mailbox/letter combination. If no such packet arrives, the RX queue is unusable in a locked state. To reenable the queue, the in-order bit in the RX CPPI Control register must be disabled by software for that queue, after which it may be enabled again if desired. The in-order mode of operation is only valid on multi-segment queues because single-segment messages will never generate RETRY responses.

Figure 20. RX CPPI Mode Explanation



In addition, multiple messages can be interleaved at the receive port due to ordering within a connected switch's output queue. This can occur when using a single or multiple priorities. The RX CPPI block must handle simultaneous interleaved multi-segment messages. This implies that state information (write pointers and srcID) must be maintained on each simultaneous message to properly store the segments in memory. The number of simultaneous transactions supported directly impacts the number of states to be stored, and the size of the buffer descriptor memory outside the peripheral. With this in mind, the peripheral's supported buffer descriptor SRAM is parameterizable. A minimum size of 1.25KB is recommended, which will allow up to 64 buffer descriptors to be stored at any given time for one core. These buffer descriptors can be configured to support any combination of single and multi-segment messages. For example, if the application only handles single-segment messages, all 64 buffers can be allotted to that queue. Note that a given RX queue can contain packets of all priorities which have been directed from any of the receive ports.

A CPU may wish to stop receiving messages and reclaim buffers belonging to a specific queue. This is called queue teardown. The CPU initiates a RX queue teardown by writing to the RX Queue Teardown command register.

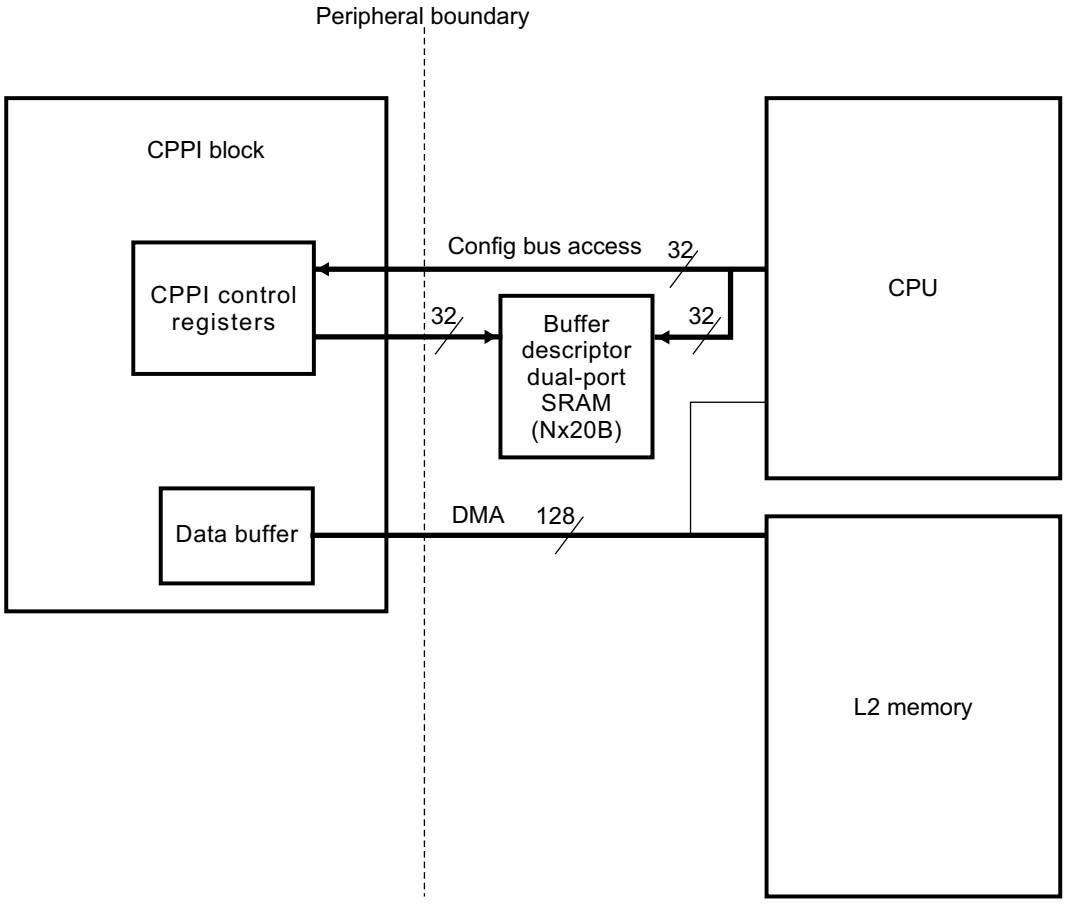
Teardown of an Rx queue causes the following actions:

- If teardown is issued by software during the time when the RX state machine is idle, then the state machine will immediately start the teardown procedure:
  - If the queue to be torn down is in-message (waiting for one or more segments), then the queue will be torn down and reported with the current buffer descriptor (teardown bit set, ownership bit cleared, CC = 100b). All other fields of the buffer descriptor are invalid. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to 0xFFFFFFFC, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is not in-message, and active (next descriptor available), then the next descriptor will be fetched and updated to report teardown (teardown bit set, ownership bit cleared, CC = 100b). All other fields of the buffer descriptor are invalid. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to 0xfffffffC, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is not in-message, but inactive (next descriptor unavailable), then no additional buffer descriptor will be written. The HDP register and the CP register remain unchanged. An interrupt is not issued. The teardown command register bit is automatically cleared by the peripheral.
- If teardown is issued by software during the time when the RXU state machine is busy, the teardown procedure will be postponed until the state machine is idle.

After the teardown process is complete and the interrupt is serviced by the CPU, the software must re-initialize the RX queue to restart normal operation.

The buffer descriptor queues are maintained in local SRAM just outside of the peripheral. This allows the quickest access time, while maintaining a level of configurability for device implementation. The SRAM is accessible by the CPU through the configuration bus. Alternatively, the buffer descriptors could use L2 memory as well.

Figure 21. CPPI Boundary Diagram



2.3.4.2 TX Operation

Outgoing messages are handled similarly, with buffer descriptor queues that are assigned by the CPUs. The queues are configured and initialized upon reset. When a CPU wants to send a message to an external RapidIO device, it writes the buffer descriptor information via the configuration bus into the SRAM. Again, there is a single buffer descriptor per RapidIO message. Upon completion of writing the buffer descriptor, the OWNERSHIP bit is set to give control to the peripheral. The CPU then writes the TX DMA State HDP register to initiate the queue transmit. For TX operation, PortID is specified to direct the outgoing packet to the appropriate port. Table 18 and Table 19 show the TX DMA state registers. Figure 22 shows the TX buffer descriptor fields. A TX buffer descriptor is a contiguous block of four 32-bit data words aligned on a 32-bit boundary.

Table 18. TX DMA State Head Descriptor Pointer (HDP) (Address Offset 0x500 – 0x53C)

Bit	Name	Description
31:0	TX Queue Head Descriptor Pointer	Tx Queue Head Descriptor Pointer: This field is the host memory address for the first buffer descriptor in the transmit queue. This field is written by the host to initiate queue transmit operations and is zeroed by the port when all packets in the queue have been transmitted. An error condition results if the host writes this field when the current field value is nonzero. The address must be 32-bit word aligned.

Table 19. TX DMA State Completion Pointer (CP) (Address Offset 0x580 – 0x5BC)

Bit	Name	Description
31:0	TX Queue Completion Pointer	Tx Queue Completion Pointer: This field is the host memory address for the transmit queue completion pointer. This register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The port uses the value written to determine if the interrupt should be deasserted.

Figure 22. TX Buffer Descriptor Fields

Word Offset	Bit Fields																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	Next Descriptor Pointer																															
1	Buffer Pointer																															
2	Dest_ID																PRI		tt		Port_ID		SSIZE					Mailbox			
3	SOP	EOP	OWNERSHIP	EOQ	TEARDOWN	Reserved											Retry_count				cc		Message Length									

Table 20. TX Buffer Descriptor Field Definitions

Field	Description
next_descriptor_pointer	Next Descriptor Pointer: The 32-bit word aligned memory address of the next buffer descriptor in the TX queue. This is the mechanism used to reference the next buffer descriptor from the current buffer descriptor. If the value of this pointer is zero then the current buffer is the last buffer in the queue. The host sets the next_descriptor_pointer.
buffer_pointer	Buffer Pointer: The byte aligned memory address of the buffer associated with the buffer descriptor. The host sets the buffer_pointer.
sop = 1	Start of Message: Indicates that the descriptor buffer is the first buffer in the message. • This bit will always be set as this device only supports one buffer per message.
eop = 1	End of Message: Indicates that the descriptor buffer is the last buffer in the message. • This bit will always be set as this device only supports one buffer per message.

Table 20. TX Buffer Descriptor Field Definitions (continued)

Field	Description
ownership	Ownership: Indicates ownership of the message and is valid only on sop. This bit is set by the host and cleared by the port when the message has been transmitted. The host uses this bit to reclaim buffers. 0: The message is owned by the host 1: The message is owned by the port
eoq	End Of Queue: Set by the port to indicate that all messages in the queue have been transmitted and the TX queue is empty. End of queue is determined by the port when the next_descriptor_pointer is zero on an eop buffer. This bit is valid only on eop. 0: The Tx queue has more messages to transfer. 1: The Descriptor buffer is the last buffer in the last message in the queue.
Teardown_Complete	Teardown Complete: Set by the port to indicate that the host commanded teardown process is complete, and the channel buffers may be reclaimed by the host. 0: The port has not completed the teardown process. 1: The port has completed the commanded teardown process.
retry_count	Message Retry Count: Set by the CPU to indicate the total number of retries allowed for this message, including all segments. Decremented by the port each time a message is retried. 000000b: Infinite Retries 000001b: Retry Message 1 time 000002b: Retry Message 2 times . . . 111111b: Retry Message 63 times
cc	Completion Code: 000: Good Completion. Message received a done response. 001: Transaction error. Message received an error response. * 010: Excessive Retries. Message received more than retry_count retry responses. 011: Transaction timeout. Transaction timer elapsed without any message response being received. 100: DMA data transfer error 101: Descriptor Programming error 110: TX Queue Teardown Complete 111: Outbound Credit not available. * An ERROR transfer completion code indicates an error in one or more segments of a transmitted multi-segment message.
message_length	Message Length: Message Length – Written by the CPU to specify the number of double-words to transmit. Message payloads are limited to a maximum size of 512 double-words (4096bytes). 000000000b: 512 double words 000000001b: 1 double word 000000010b: 2 double words . . . 111111111b: 511 double words
Dest_id	Destination Node Id: Unique Node identifier for the Destination of the message.
pri	Message Priority: Specifies the SRIO priority at which the message will be sent. Messages should not be sent at a priority level of 3 because the message response is required to promote the priority to avoid system deadlock. It is the responsibility of the software to assign the appropriate outgoing priority.

Table 20. TX Buffer Descriptor Field Definitions (continued)

Field	Description
tt	RapidIO *tt* field specifying 8- or 16-bit DeviceIDs 00: 8b deviceIDs 01: 16b deviceIDs 10: reserved 11: reserved
PortID	Port number for routing outgoing packet.
SSIZE	RIO standard message payload size. Indicates how the hardware should segment the outgoing message by specifying the maximum number of double-words per packet. If the message is a multi-segment message, this field remains the same for all outgoing segments. All segments of the message, except for the last segment, have payloads equal to this size. The last message segment may be equal or less than this size. Maximum message size for a 16 segment message is shown below. Message_length/16 must be less than or equal to Ssize, if not, the message is not sent and CC 101b is set. 0000b - 1000b: Reserved 1001b: 1 Double-word payload (Supports up to a 128B message) 1010b: 2 Double-word payload (Supports up to a 256B message) 1011b: 4 Double-word payload (Supports up to a 512B message) 1100b: 8 Double-word payload (Supports up to a 1024B message) 1101b: 16 Double-word payload (Supports up to a 2048B message) 1110b: 32 Double-word payload (Supports up to a 4096B message) 1111b: Reserved
mailbox	Destination Mailbox: Specifies the mailbox to which the message will be sent. 000000b: Mailbox 0 000001b: Mailbox 1 . . . 000100b: Mailbox 4 . . . 111111b: Mailbox 63 For multi-segment messages, only the 2 LSBs of this mailbox field are valid. Hardware will ignore the 4 MSBs of this field if the outgoing message is multi-segment.

Once the port controls the buffer descriptor, the DEST_ID field can be queried to determine flow control. If the transaction has been flow controlled, the DMA bus READ request is postponed so that the TX buffer space is not wasted. Because buffer descriptors cannot be reordered in the link list, if the transaction at the head of the buffer descriptor queue is flow controlled, HOL blocking will occur on that queue. When this occurs, all transactions located in that queue are stalled. To counter the affects and reduce back-up of more TX packets, multiple queues are available. The peripheral supports a total of 16 assignable TX queues and their associated TX DMA state registers. The transmission order between queues is based on programmable weighted round-robin at the message level. The programmable registers are shown in Figure 23. This scheme allows configurability of the queue transmission order, as well as the weight of each queue within the round robin. Upon enabling the peripheral with the Peripheral Control Register (0x0004), the TX state machine begins by processing the TX_Queue_Map0. It will attempt to process the queue and number of buffer descriptors from that queue programmed in this mapper. Then it will move to TX_Queue_Map1, followed by TX_Queue_Map2 and so forth. It is important to note that this mapping order is fixed in a circular pattern. Each mapper can point to any queue and multiple mappers can point to a single queue. If a mapper points to an in-active queue, the peripheral recognizes this and moves to the next mapper. In order for an active queue to transmit packets, at least one mapper must be pointing to that queue. The default settings dictate an equally weighted round robin that starts on queue0 and increments by one until reaching queue15. If the application requires precise control over the order of data sent out of the device after power up or reset, you should program the map control registers, set up the TX queues and initialize the HDP, and finally enable the peripheral with the Peripheral Control Register.

Figure 23. Weighted Round Robin Programming Registers (Address Offset 0x7E0 – 0x7EC)

TX_QUEUE_CNTL0- Address Offset (0x7E0)

31 – 24	23 – 16
TX_Queue_Map3	TX_Queue_Map2

15 – 8	7 – 0
TX_Queue_Map1	TX_Queue_Map0

TX_QUEUE_CNTL1- Address Offset (0x7E4)

31 – 24	23 – 16
TX_Queue_Map7	TX_Queue_Map6

15 – 8	7 – 0
TX_Queue_Map5	TX_Queue_Map4

TX_QUEUE_CNTL2- Address Offset (0x7E8)

31 – 24	23 – 16
TX_Queue_Map11	TX_Queue_Map10

15 – 8	7 – 0
TX_Queue_Map9	TX_Queue_Map8

TX_QUEUE_CNTL3- Address Offset (0x7EC)

31 – 24	23 – 16
TX_Queue_Map15	TX_Queue_Map14

15 – 8	7 – 0
TX_Queue_Map13	TX_Queue_Map12

Table 21. Weighted Round Robin Programming Registers (Address Offset 0x7E0 – 0x7EC)

Name	Bit	Access	Reset Value	Description
TX_Queue_Map0	[7:0]	R/W	0x00	[7:4] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map1 [3:0] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map1	[15:8]	R/W	0x01	15:12] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map2 [11:8] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map2	[23:16]	R/W	0x02	[23:20] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map3 [19:16] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map3	[31:24]	R/W	0x03	[31:28] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map4 [27:24] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map4	[7:0]	R/W	0x04	[7:4] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map5 [3:0] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map5	[15:8]	R/W	0x05	[15:12] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map6 [11:8] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map6	[23:16]	R/W	0x06	[23:20] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map7 [19:16] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map7	[31:24]	R/W	0x07	[31:28] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map8 [27:24] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map8	[7:0]	R/W	0x08	[7:4] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map9 [3:0] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map9	[15:8]	R/W	0x09	[15:12] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map10 [11:8] = Pointer to a Queue, programmable to any of the 16 TX queues

Table 21. Weighted Round Robin Programming Registers (Address Offset 0x7E0 – 0x7EC) (continued)

Name	Bit	Access	Reset Value	Description
TX_Queue_Map10	[23:16]	R/W	0x0A	[23:20] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map11 [19:16] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map11	[31:24]	R/W	0x0B	[31:28] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map12 [27:24] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map12	[7:0]	R/W	0x0C	[7:4] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map13 [3:0] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map13	[15:8]	R/W	0x0D	[15:12] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map14 [11:8] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map14	[23:16]	R/W	0x0E	[23:20] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map15 [19:16] = Pointer to a Queue, programmable to any of the 16 TX queues
TX_Queue_Map15	[31:24]	R/W	0x0F	[31:28] = Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map0 [27:24] = Pointer to a Queue, programmable to any of the 16 TX queues

The TX queues are treated differently than the RX queues. A TX queue can mix single and multi-segment message buffer descriptors. The software manages the queue usage.

All outgoing message segments have responses that indicate the status of the transaction. Responses may indicate DONE, ERROR or RETRY. A buffer descriptor may be released back to CPU control (OWNERSHIP = 0), only after all segment responses are received, or alternatively if a response timeout occurs. Timeouts and response evaluation have high priority in the state-machine since they are the only means to release TX packet resources. The CC is set in the buffer descriptor to indicate the response status to the CPU. If there is a RETRY response, the TX CPPI module will immediately retry the packet before continuing to the next queue in the round-robin, as long as the RETRY_COUNT is not exceeded. Once this limit is exceeded, the buffer can be released back to CPU control with the appropriate CC set. Retry of a message segment does not imply retrying a whole message. Only segments for which a RETRY response is received should be re-transmitted. This will involve calculating the correct starting point within the TX data buffer based on the failed segment number and message length. To achieve respectable performance, the peripheral must not wait for a message/segment response before sending out the next packet.

Since RapidIO allows for out-of-order responses, the TX CPPI hardware must support this functionality. As responses are received, the hardware updates the corresponding TX buffer descriptor to reflect the status. However, if the response is out-of-order, the hardware does not update the CP or set the corresponding interrupt. Only after all preceding outstanding message responses are received, will the CP and interrupt be updated. This ensures that a contiguous block of buffer descriptors, starting at the oldest outstanding descriptor, has been processed by the hardware and is ready for the CPU to reclaim the buffers.

A transaction timeout is used by all outgoing message and directIO packets. It is defined by the 24-bit value in the Port Response Time-out CSR. The RapidIO specification states that the maximum time interval (all 1s) is between 3 and 6 seconds. A logical layer timeout occurs if the response packet is not received before a countdown timer (initialized to this CSR value) reaches zero. Since transaction responses can be acknowledged out-of-order, a timer is needed for each supported outstanding packet in the TX queue. Each outstanding packet response timer requires a 4-bit register. The register is loaded with the current timecode when the transaction is sent. Each time the timecode changes, a 4-bit compare is done to the 16 outstanding packet registers. If the register becomes equal to the timecode again, without a response being seen, then the transaction has timed out and the buffer descriptor is written.

Essentially, instead of the 24-bit value representing the period of the response timer, the period is now defined as P = (2^24 x 16)/F. This means the countdown timer frequency needs to be 44.7 – 89.5Mhz for a 6 – 3 second response timeout. Since the needed timer frequency is derived from the DMA bus clock (which is device dependent), the hardware supports a programmable configuration register field to properly scale the clock frequency. This configuration register field is described in the Peripheral Control Register (Address offset 0x0004).

The CPU initiates a TX queue teardown by writing to the TX Queue Teardown command register. Teardown of a TX queue will cause the following actions:

- No new messages will be sent
- All messages (single and multi-segment) already started will be completed
  - Failing to complete the message TX would leave an active receiver blocked waiting for the final segments until the transaction eventually times-out.
  - Note that normal Tx SM operation is to not send any more segments once an error response has been received on any segment. So if the receiver has been torn-down (and is receiving error responses) multi-segment transmit will complete as soon as all in-transit segments have been responded to.
- When all in-transit messages/segments have been responded to, teardown will be completed as follows:
  - If the queue is active, the teardown bit will be set in the next buffer descriptor in the queue. The peripheral completes the teardown procedure by clearing the HDP register, setting the CP register to 0xfffffffC, and issuing an interrupt for the given queue. The teardown command register bit is automatically cleared by the peripheral.
  - If the queue is in-active (no additional buffer descriptors available), or becomes inactive after a message in transmission is completed, no buffer descriptor fields are written. The HDP register and the CP register remain unchanged. An interrupt is not issued. The teardown command register bit is automatically cleared by the peripheral.
  - Because of topology differences between flow's response, packets may arrive in a different order to the order of requests.

After the teardown process is complete and the interrupt is serviced by the CPU, software must re-initialize the TX queue to restart normal operation.

2.3.4.3 Detailed Data Path Description

The CPPI module is the message passing protocol engine of the RapidIO peripheral. Messages contain application specific data that is pushed to the receiving device comparable to a streaming write. Messages do not contain read operations, but do have response packets.

The data path for this module uses the DMA bus as the DMA interface. The ftype header field of the received RapidIO message packets are decoded by the logical layer of the peripheral. Only Type 11 and Type 13 (transaction type1) packets are routed to this module. Data is routed from the priority based RX FIFOs to the CPPI module's data buffer within the shared buffer pool. The mbox header fields are examined by the MailBox Mapper block of the CPPI module. Based on the mailbox, and message length, the data is assigned memory addresses within memory. Data is transferred via DMA bus commands to memory from the buffer space of the peripheral. The maximum buffer space should accommodate 256B of data, as that is the maximum payload size of a RapidIO packet. Each message in memory will be represented by a buffer descriptor in the queue.

2.3.4.4 Reset and Power Down State

Upon reset, the CPPI module must be configured by the CPU. The CPU sets up the receive and transmit queues in memory. Then the CPU updates the CPPI module with the appropriate Rx/TX DMA state head descriptor pointer, so the peripheral knows with which buffer descriptor address to start. Additionally, the CPU must provide the CPPI module with initial buffer descriptor values for each data buffer. This step is described more extensively in Section 2.3.6 of the CPPI specification.

The CPPI module can be powered down if the message passing protocol is not being supported in the application. For example, if the direct I/O protocol is being used for data transfers, powering down the CPPI module will save power. In this situation, the buffer descriptor queue SRAMs and mailbox mapper logic should be powered down. Clocks should be gated to these blocks while in the power down state. Section 2.3.9 describes this in detail.

2.3.4.5 Message Passing Software Requirements

Software performs the following functions for messaging:

RX Operation

- Assigns Mailbox-to-queue mapping and allowable SourceIDs/mailbox- Queue Mapping
- Sets up associated buffer descriptor memory – CPPI RAM or L2 RAM
- Link-lists the buffer descriptors, next_descriptor_pointer
- Assigns single segment (256B payload) and multi-segment (4KB payload) buffers to queues buffer_length
- Assigns buffer descriptor to data buffer, buffer_pointer
- Gives control of the buffer to the peripheral, ownership = 1

Configures and initiates RX queues

- Assigns Head Descriptor Pointer, HDP, for up to 16 queues: RX DMA State HDP
- Port begins to consume buffers beginning with HDP descriptor and sets ownership = 0 for each buffer descriptor used. Writes Completion Pointer, CP, RX DMA State CP and moves to next buffer.
- Port hardware generates pending interrupt when CP is written. Physical interrupt generated when Interrupt Pacing Count down timer = 0.

Processes interrupt

- Determines ICSR bit and process corresponding queue until ownership = 1 or eoq = 1
- Sets processed buffer descriptor ownership = 1
- Writes CP value of last buffer descriptor processed
- Port hardware clears ICSR bit only if the CP value written by CPU equals port written value in the RX DMA State CP register
- Resets interrupt pacing value

TX Operation

Sets up associated buffer descriptor memory – CPPI RAM or L2 RAM

- Link-lists the buffer descriptors, next_descriptor_pointer
- Assigns buffer descriptor to data buffer, buffer_pointer
- Gives control of the buffer to the CPU, ownership = 0
- CPU writes buffer descriptors beginning with HDP descriptor and sets ownership = 1 for each used
- Specifies RIO fields: Dest_id, Pri, tt, Mailbox
- Sets parameters: PortID, Message_length
- Port starts queue transmit on CPU write to HDP for up to 16 queues - TX DMA State HDP
- Port processes corresponding queues until ownership = 0 or next_descriptor_pointer = all 0s. Port sets eoq = 1 and writes all 0s to the HDP.
- When each packet transmission is complete, the port sets ownership = 0 and issues an interrupt to the CPU by writing the last processed buffer descriptor address to the CP, TX DMA State CP

Processes interrupt

- The CPU processes the buffer queue to reclaim buffers. If ownership = 0, the packet has been transmitted and the buffer is reclaimed.
- CPU processes the queue until eoq = 1 or ownership = 1
- CPU determines all packets have been transmitted if ownership = 0, eoq = 1, and next_descriptor_pointer = all 0s in last processed buffer descriptor
- CPU acknowledges the interrupt after re-claiming all available buffer descriptors.
- CPU acknowledges the interrupt by writing the CP value

- This value is compared against the port written value in the TX DMA State CP register, if equal, the interrupt is deasserted.

Initialization Example

```
SRIO_REGS->Queue0_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue1_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue2_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue3_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue4_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue5_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue6_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue7_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue8_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue9_RXDMA_HDP   = 0 ;
SRIO_REGS->Queue10_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue11_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue12_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue13_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue14_RXDMA_HDP  = 0 ;
SRIO_REGS->Queue15_RXDMA_HDP  = 0 ;
```

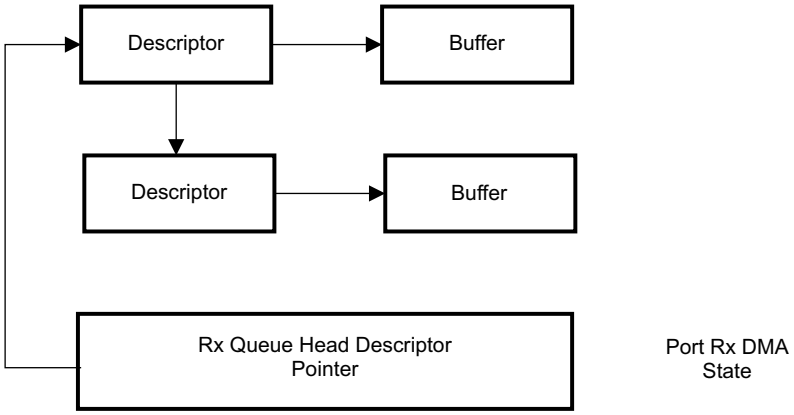
Queue Mapping

```
SRIO_REGS->RXU_MAP01_L = CSL_FMK( SRIO_RXU_MAP01_L_LETTER_MASK, 3)|
                         CSL_FMK( SRIO_RXU_MAP01_L_MAILBOX_MASK, 0x3F)|
                         CSL_FMK( SRIO_RXU_MAP01_L_LETTER, 0)|
                         CSL_FMK( SRIO_RXU_MAP01_L_MAILBOX, 1)|
                         CSL_FMK( SRIO_RXU_MAP01_L_SOURCEID, 0xBEEF);
SRIO_REGS->RXU_MAP01_H = CSL_FMK( SRIO_RXU_MAP01_H_TT, 1)|
                         CSL_FMK( SRIO_RXU_MAP01_H_QUEUE_ID, 0)|
                         CSL_FMK( SRIO_RXU_MAP01_H_PROMISCUOUS, 1)|
                         CSL_FMK( SRIO_RXU_MAP01_H_SEGMENT_MAPPING, 1);
```

RX Buffer Descriptor

```
RX_DESCP0_0->RXDESC0 = CSL_FMK( SRIO_RXDESC0_N_POINTER,(int )RX_DESCP0_1 );
                     //link to RX_DESCP0_1
                     //poll mode, extended address type 2,5,6
RX_DESCP0_0->RXDESC1 = CSL_FMK( SRIO_RXDESC1_B_POINTER,(int )&rcvBuff1[0] );
                     //32bit = type 2,5,6. 24bit = type 8
RX_DESCP0_0->RXDESC2 = CSL_FMK( SRIO_RXDESC2_SRC_ID, 0xBEEF)|
                       CSL_FMK( SRIO_RXDESC2_PRI, 1)|
                       CSL_FMK( SRIO_RXDESC2_TT, 1)|
                       CSL_FMK( SRIO_RXDESC2_MAILBOX, 0);
RX_DESCP0_0->RXDESC3 = CSL_FMK( SRIO_RXDESC3_SOP,1 )|
                       CSL_FMK( SRIO_RXDESC3_EOP,1 )|
                       CSL_FMK( SRIO_RXDESC3_OWNERSHIP,1 )|
                       CSL_FMK( SRIO_RXDESC3_EOQ,1 )|
                       CSL_FMK( SRIO_RXDESC3_TEARDOWN,0 )|
                       CSL_FMK( SRIO_RXDESC3_CC,0 )|
                       CSL_FMK(SRIO_RXDESC3_MESSAGE_LENGTH,MLEN_512DW);
RX_DESCP0_1->RXDESC0 = CSL_FMK( SRIO_RXDESC0_N_POINTER, 0);
                     //end of message
                     //poll mode, extended address type 2,5,6
RX_DESCP0_1->RXDESC1 = CSL_FMK( SRIO_RXDESC1_B_POINTER,(int )&rcvBuff2[0] );
                     //32bit = type 2,5,6. 24bit = type 8
RX_DESCP0_1->RXDESC2 = CSL_FMK( SRIO_RXDESC2_SRC_ID, 0xBEEF)|
                       CSL_FMK( SRIO_RXDESC2_PRI, 1)|
                       CSL_FMK( SRIO_RXDESC2_TT, 1)|
                       CSL_FMK( SRIO_RXDESC2_MAILBOX, 1);
RX_DESCP0_1->RXDESC3 = CSL_FMK( SRIO_RXDESC3_SOP,1 )|
                       CSL_FMK( SRIO_RXDESC3_EOP,1 )|
                       CSL_FMK( SRIO_RXDESC3_OWNERSHIP,1 )|
                       CSL_FMK( SRIO_RXDESC3_EOQ,1 )|
                       CSL_FMK( SRIO_RXDESC3_TEARDOWN,0 )|
                       CSL_FMK( SRIO_RXDESC3_CC,0 )|
                       CSL_FMK( SRIO_RXDESC3_MESSAGE_LENGTH,MLEN_512DW );
```

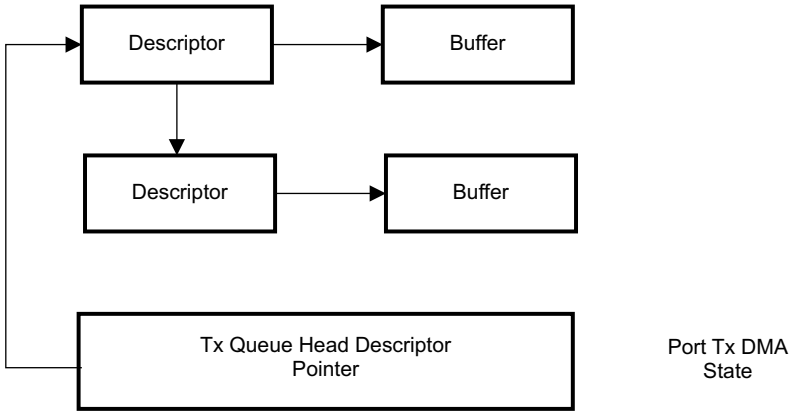
Figure 24. RX Buffer Descriptor



TX Buffer Descriptor

```
TX_DESCP0_0->TXDESC0 =  CSL_FMK( SRIO_TXDESC0_N_POINTER,(int )TX_DESCP0_1 );
                       //link to TX_DESCP0_1
                       //NDP
TX_DESCP0_0->TXDESC1 =  CSL_FMK( SRIO_TXDESC1_B_POINTER,(int )&xmtBuff1[0] );
                       //Buffer Pointer
TX_DESCP0_0->TXDESC2 =  CSL_FMK( SRIO_TXDESC2_DESTID, 0xBEEF)|
                        CSL_FMK( SRIO_TXDESC2_PRI, 1)|
                        CSL_FMK( SRIO_TXDESC2_TT, 1)|
                        CSL_FMK( SRIO_TXDESC2_PORTID, 3)|
                        CSL_FMK( SRIO_TXDESC2_SSIZE, SSIZE_256B)|
                        CSL_FMK( SRIO_TXDESC2_MAILBOX, 0);
TX_DESCP0_0->TXDESC3 =  CSL_FMK( SRIO_TXDESC3_SOP,1 )|
                        CSL_FMK( SRIO_TXDESC3_EOP,1 )|
                        CSL_FMK( SRIO_TXDESC3_OWNERSHIP,1 )|
                        CSL_FMK( SRIO_TXDESC3_EOQ,1 )|
                        CSL_FMK( SRIO_TXDESC3_TEARDOWN,0 )|
                        CSL_FMK( SRIO_TXDESC3_RETRY_COUNT,0 )|
                        CSL_FMK( SRIO_TXDESC3_CC,0 )|
                        CSL_FMK( SRIO_TXDESC3_MESSAGE_LENGTH,MLEN_512DW );
TX_DESCP0_1->TXDESC0 =  CSL_FMK( SRIO_TXDESC0_N_POINTER, 0);
                       //end of message
                       //poll mode, extended address type 2,5,6
TX_DESCP0_1->TXDESC1 =  CSL_FMK( SRIO_TXDESC1_B_POINTER,(int )&xmtBuff2[0] );
                       //32bit = type 2,5,6. 24bit = type 8
TX_DESCP0_1->TXDESC2 =  CSL_FMK( SRIO_TXDESC2_DESTID, 0xBEEF)|
                        CSL_FMK( SRIO_TXDESC2_PRI, 1)|
                        CSL_FMK( SRIO_TXDESC2_TT, 1)|
                        CSL_FMK( SRIO_TXDESC2_PORTID, 3)|
                        CSL_FMK( SRIO_TXDESC2_SSIZE, SSIZE_256B)|
                        CSL_FMK( SRIO_TXDESC2_MAILBOX, 1);
TX_DESCP0_1->TXDESC3 =  CSL_FMK( SRIO_TXDESC3_SOP,1 )|
                        CSL_FMK( SRIO_TXDESC3_EOP,1 )|
                        CSL_FMK( SRIO_TXDESC3_OWNERSHIP,1 )|
                        CSL_FMK( SRIO_TXDESC3_EOQ,1 )|
                        CSL_FMK( SRIO_TXDESC3_TEARDOWN,0 )|
                        CSL_FMK( SRIO_TXDESC3_RETRY_COUNT,0 )|
                        CSL_FMK( SRIO_TXDESC3_CC,0 )|
                        CSL_FMK( SRIO_TXDESC3_MESSAGE_LENGTH,MLEN_512DW );
```

Figure 25. TX Buffer Descriptor



Start Message Passing

```
SRIO_REGS->Queue0_RXDMA_HDP     = (int )RX_DESCP0_0 ;
SRIO_REGS->Queue0_TxDMA_HDP     = (int )TX_DESCP0_0 ;
```

2.3.5 Maintenance

The type 8 MAINTENANCE packet format accesses the RapidIO capability registers (CARs), command and status registers (CSRs), and data structures. Unlike other request formats, the type 8 packet format serves as both the request and the response format for maintenance operations. Type 8 packets contain no addresses and only contain data payloads for write requests and read responses. All configuration register read accesses are word (4-byte) accesses. All configuration register write accesses are also word (4-byte) accesses.

The wrsize field specifies the maximum size of the data payload for multiple double-word transactions. The data payload may not exceed that size but may be smaller if desired. Both the maintenance read and the maintenance write request generate the appropriate maintenance response.

The maintenance port-write operation is a write operation that does not have guaranteed delivery and does not have an associated response. This maintenance operation is useful for sending messages such as error indicators or status information from a device that does not contain an endpoint, such as a switch. The data payload is typically placed in a queue in the targeted endpoint and an interrupt is typically generated to a local processor. A port-write request to a queue that is full or busy servicing another request may be discarded.

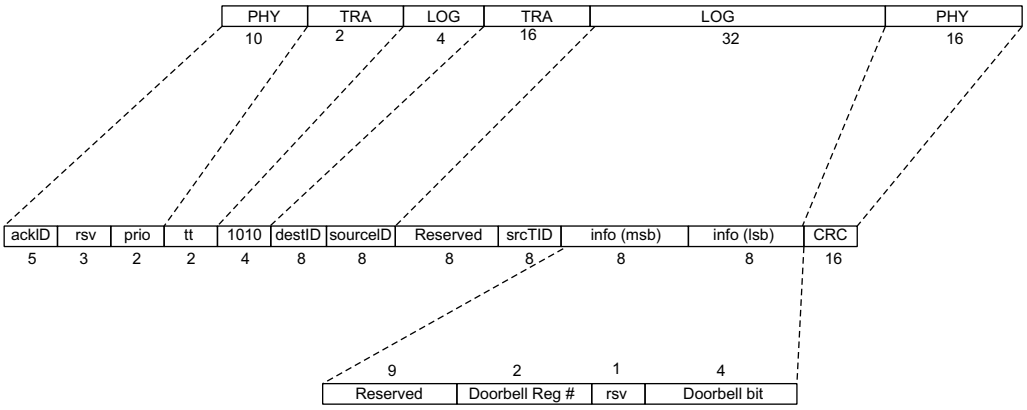
```
SRIO_REGS->LSU1_Reg0 =  CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
SRIO_REGS->LSU1_Reg1 =  CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET, (int )car_csr );
SRIO_REGS->LSU1_Reg2 =  CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, (int )&xmtBuff[0]);
SRIO_REGS->LSU1_Reg3 =  CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT,byte_count );
                        //=4
SRIO_REGS->LSU1_Reg4 =  CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,0 )|
                        //0b00
                        CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )|
                        //0b00
                        CSL_FMK( SRIO_LSU1_REG4_XAMBS,0 )|
                        //no extended address
                        CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )|
                        //tt = 0b01
                        CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )|
                        //0xBEEF
                        CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,0 );
                        //0 = event-driven, 1 = poll
SRIO_REGS->LSU1_Reg5 =  CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                        CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x03 )|
                        //hop = 0x03
                        CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );
                        //type = REQ_MAINT_RD
```

2.3.6 Doorbell

The doorbell operation, consisting of the DOORBELL and RESPONSE transactions (typically a DONE response), as shown in Figure 26, is used by a processing element to send a very short message to another processing element through the interconnect fabric. The DOORBELL transaction contains the info field to hold information and does not have a data payload. This field is software-defined and can be used for any desired purpose; see Section 3.1.4, *Type 10 Packet Formats (Doorbell Class)*, for information about the info field. A processing element that receives a doorbell transaction takes the packet and puts it in a doorbell message queue within the processing element. This queue may be implemented in hardware or in local memory. This behavior is similar to that of typical message passing mailbox hardware. The local processor is expected to read the queue to determine the sending processing element and the info field, and determine what action to take.

The DOORBELL functionality is user-defined, but this packet type is commonly used to initiate CPU interrupts. A DOORBELL packet is not associated with a particular data packet that was previously transferred, so the info field of the packet must be configured to reflect the DOORBELL bit to be serviced for the correct TID info to be processed.

Figure 26. Doorbell Operation



The DOORBELL packet's 16-bit INFO field indicates which DOORBELL register interrupt bit to set. There are four DOORBELL registers, each currently with 16 bits, allowing 64 interrupt sources or circular buffers. Each bit can be assigned to any core as described below by the Interrupt Condition Routing Registers. Additionally, each status bit is user-defined for the application. For instance, it may be desirable to support multiple priorities with multiple TID circular buffers per core if control data uses a high priority (i.e., priority = 2), while data packets are sent on priority 0 or 1. This allows the control packets to have preference in the switch fabric and arrive as quickly as possible. Since it may be required to interrupt the CPU for both data and control packet processing separately, separate circular buffers are used, and DOORBELL packets need to distinguish between them for interrupt servicing. If any reserved bit in the DOORBELL info field is set, an error response is sent.

```
SRIO_REGS->LSU1_Reg0 =  CSL_FMK( SRIO_LSU1_REG0_RAPIDIO_ADDRESS_MSB,0 );
SRIO_REGS->LSU1_Reg1 =  CSL_FMK( SRIO_LSU1_REG1_ADDRESS_LSB_CONFIG_OFFSET, 0);
SRIO_REGS->LSU1_Reg2 =  CSL_FMK( SRIO_LSU1_REG2_DSP_ADDRESS, 0);
SRIO_REGS->LSU1_Reg3 =  CSL_FMK( SRIO_LSU1_REG3_BYTE_COUNT, 0 );
SRIO_REGS->LSU1_Reg4 =  CSL_FMK( SRIO_LSU1_REG4_OUTPORTID,1 )|
                        CSL_FMK( SRIO_LSU1_REG4_PRIORITY,0 )|
                        CSL_FMK( SRIO_LSU1_REG4_XAMBS,0 )|
                        //no extended address
                        CSL_FMK( SRIO_LSU1_REG4_ID_SIZE,1 )|
                        //tt = 0b01
                        CSL_FMK( SRIO_LSU1_REG4_DESTID,0xBEEF )|
                        CSL_FMK( SRIO_LSU1_REG4_INTERRUPT_REQ,0 );
                        //0 = event-driven, 1 = poll
SRIO_REGS->LSU1_Reg5 =  CSL_FMK( SRIO_LSU1_REG5_DRBLL_INFO,0x0000 )|
                        CSL_FMK( SRIO_LSU1_REG5_HOP_COUNT,0x03 )|
                        //hop = 0x03
                        CSL_FMK( SRIO_LSU1_REG5_PACKET_TYPE,type );
                        //type = REQ_DOORBELL
```

2.3.7 Congestion Control

The RapidIO Flow Control specification is referenced in Table 1. This section describes the requirements and implementation of congestion control within the peripheral.

The peripheral is notified of switch fabric congestion through type 7 RapidIO packets. The packets are referred to as Congestion Control Packets (CCPs). The purpose of these packets is to turn off (Xoff), or turn on (Xon) specific flows defined by DESTID and PRIORITY of outgoing packets. CCPs are sent at the highest priority in an attempt to address fabric congestion as quickly as possible. CCPs do not have a response packet and they do not have guaranteed delivery.

When the peripheral receives an Xoff CCP, the peripheral must block outgoing LSU and CPPI packets that are destined for that flow. When the peripheral receives an Xon, the flow may be enabled. Since CCPs may arrive from different switches within the fabric, it is possible to receive multiple Xoff CCPs for the same flow. For this reason, the peripheral must maintain a table and count of Xoff CCPs for each flow. For example, if two Xoff CCPs are received for a given flow, two Xon CCPs must be received before the flow is enabled.

Since CCPs do not have guaranteed delivery and can be dropped by the fabric, an implicit method of enabling an Xoff'd flow must exist. A simple timeout method is used. Additionally, flow control checks can be enabled or disabled through the Transmit Source Flow Control Masks. Received CCPs are not passed through the DMA bus interface.

2.3.7.1 Detailed Description

To avoid large and complex table management, a basic scheme is implemented for RIO congestion management. The primary goal is to avoid large parallel searches of a centralized congested route table for each outgoing packet request. The congested route table requirements and subsequent searches would be overwhelming if each possible DESTID and PRIORITY combination had its own entry. To implement a more basic scheme, the following assumptions have been made:

- A small number of flows constitute the majority of traffic, and these flows are most likely to cause congestion
- HOL blocking is undesired, but allowable for TX CPPI queues
- Flow control will be based on DESTID only, regardless of PRIORITY

The congested route table is therefore more static in nature. Instead of dynamically updating a table with each CCP's flow information as it arrives, a small finite-entry table is set up and configured by software to reflect the more critical flows it is using. Only these flows have a discrete table entry. A 16 entry table reflects 15 critical flows, leaving the sixteenth entry for general other flows, which are categorized together. Figure 27 shows the MMR table entries that are programmable by the CPU through the configuration bus. A 3-bit hardware counter is implemented for table entries 0 through 14, to maintain a count of Xoff CCPs for that flow. The other flows table entry counts Xoff CCPs for all flows other than the discrete entries. The counter for this table entry is 5-bit. All outgoing flows with non-zero Xoff counts are disabled. The counter value is decremented for each corresponding Xon CCP that is received, but it should not decrement below zero. Additionally, a hardware timer is needed for each table entry to turn on flows that may have been abandoned by lost Xon CCPs. The timer value needs to be an order of magnitude larger than the 32b Port Response Time-out CSR value. For this reason, each transmission source will add 2 bits to its 4-bit response time-out counter described in Section 2.3.3.3 and Section 2.3.4.2. The additional 2 bits count three timecode revolutions and provide an implicit Xon timer equal to 3X the Response time-out counter value.

Figure 27. Flow Control Table Entry Registers (Address Offset 0x0900 - 0x093C)

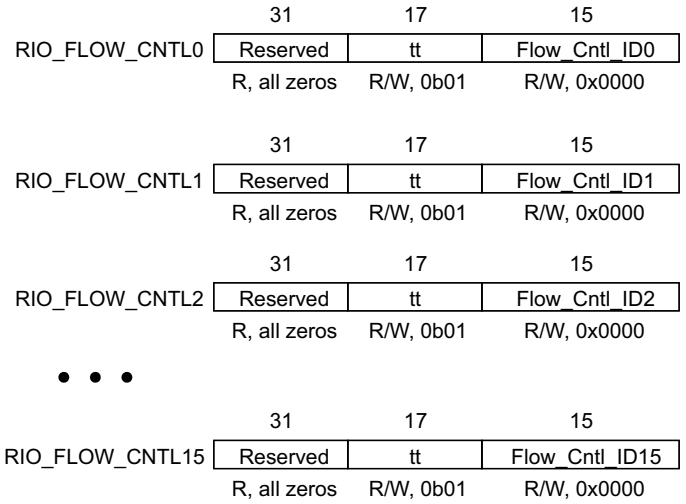


Table 22. Flow Control Table Entry Registers (Address Offset 0x0900 - 0x093C)

Name	Bit	Access	Reset Value	Description
tt	[17:16]	R/W	01b	Selects Flow_Cntl_ID length. 00b – 8b IDs 01b – 16b IDs 10b – 11b - reserved
Flow_Cntl_ID0	[15:0]	R/W	0x0000	DestID of Flow 0
Flow_Cntl_ID1	[15:0]	R/W	0x0000	DestID of Flow 1
Flow_Cntl_ID2	[15:0]	R/W	0x0000	DestID of Flow 2
Flow_Cntl_ID3	[15:0]	R/W	0x0000	DestID of Flow 3
Flow_Cntl_ID4	[15:0]	R/W	0x0000	DestID of Flow 4
Flow_Cntl_ID5	[15:0]	R/W	0x0000	DestID of Flow 5
Flow_Cntl_ID6	[15:0]	R/W	0x0000	DestID of Flow 6
Flow_Cntl_ID7	[15:0]	R/W	0x0000	DestID of Flow 7
Flow_Cntl_ID8	[15:0]	R/W	0x0000	DestID of Flow 8
Flow_Cntl_ID9	[15:0]	R/W	0x0000	DestID of Flow 9
Flow_Cntl_ID10	[15:0]	R/W	0x0000	DestID of Flow 10
Flow_Cntl_ID11	[15:0]	R/W	0x0000	DestID of Flow 11
Flow_Cntl_ID12	[15:0]	R/W	0x0000	DestID of Flow 12
Flow_Cntl_ID13	[15:0]	R/W	0x0000	DestID of Flow 13
Flow_Cntl_ID14	[15:0]	R/W	0x0000	DestID of Flow 14
Flow_Cntl_ID15	[15:0]	R/W	0x0000	DestID of Flow 15, if all 0s this table entry represents all flows other than flows 0 - 14

Each transmit source, including LSU or Tx CPPI queues, indicates which of the 16 flows it uses by having a mask register. Figure 28 illustrates the required 16-bit registers with bit masks. The CPU must configure these registers upon reset. The default setting is all 1s, indicating that the transmit source supports all flows. If the register is set to all 0s, the transmit source communicates that it does not support any flow, and consequently, that source is never flow-controlled. If any of the table entry counters that a transmit source supports have a corresponding non-zero Xoff count, the transmit source is flow-controlled. A simple 16-bit bus indicates the Xoff state of all 16 flows and is compared to the transmit source mask register. Each source interprets this result and performs flow control accordingly. For example, an LSU module that is flow-controlled can reload its registers and attempt to send a packet to another flow, while a TX CPPI queue that is flow-controlled may create HOL blocking issues on that queue.

Figure 28. Transmit Source Flow Control Masks

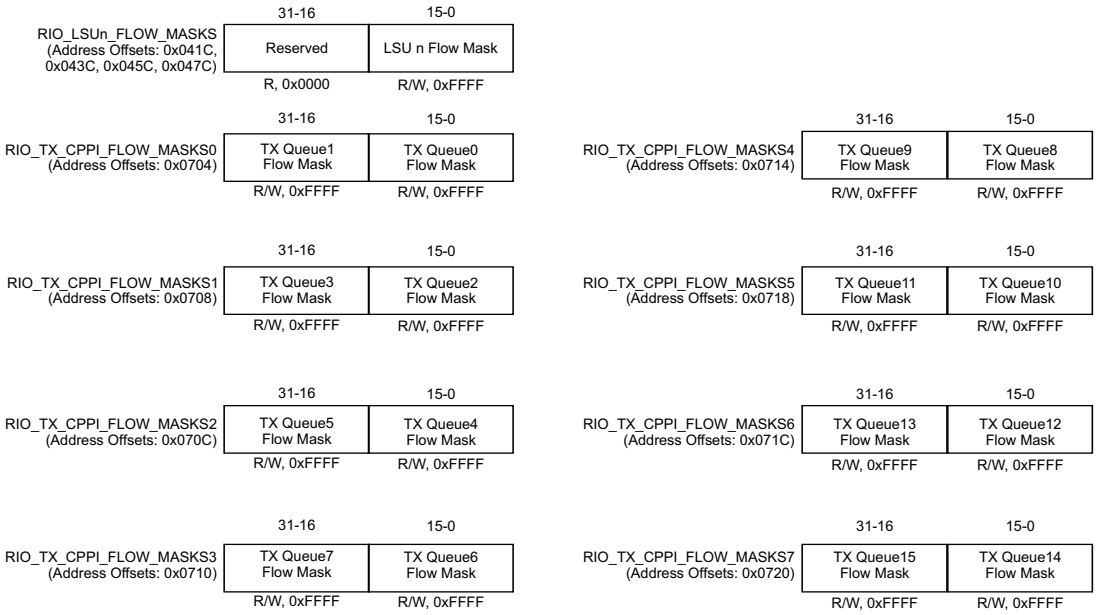


Table 23. Transmit Source Flow Control Masks

Name	Bit	Access	Reset Value	Description
Flow Mask	0	R/W	1b	0b – TX source doesn't support Flow0 from table entry 1b – TX source does support Flow0 from table entry
Flow Mask	1	R/W	1b	0b – TX source doesn't support Flow1 from table entry 1b – TX source does support Flow1 from table entry
Flow Mask	2	R/W	1b	0b – TX source doesn't support Flow2 from table entry 1b – TX source does support Flow2 from table entry
Flow Mask	3	R/W	1b	0b – TX source doesn't support Flow3 from table entry 1b – TX source does support Flow3 from table entry
Flow Mask	4	R/W	1b	0b – TX source doesn't support Flow4 from table entry 1b – TX source does support Flow4 from table entry
Flow Mask	5	R/W	1b	0b – TX source doesn't support Flow5 from table entry 1b – TX source does support Flow5 from table entry
Flow Mask	6	R/W	1b	0b – TX source doesn't support Flow6 from table entry 1b – TX source does support Flow6 from table entry
Flow Mask	7	R/W	1b	0b – TX source doesn't support Flow7 from table entry 1b – TX source does support Flow7 from table entry
Flow Mask	8	R/W	1b	0b – TX source doesn't support Flow8 from table entry 1b – TX source does support Flow8 from table entry
Flow Mask	9	R/W	1b	0b – TX source doesn't support Flow9 from table entry 1b – TX source does support Flow9 from table entry
Flow Mask	10	R/W	1b	0b – TX source doesn't support Flow10 from table entry 1b – TX source does support Flow10 from table entry
Flow Mask	11	R/W	1b	0b – TX source doesn't support Flow11 from table entry 1b – TX source does support Flow11 from table entry
Flow Mask	12	R/W	1b	0b – TX source doesn't support Flow12 from table entry 1b – TX source does support Flow12 from table entry
Flow Mask	13	R/W	1b	0b – TX source doesn't support Flow13 from table entry 1b – TX source does support Flow13 from table entry
Flow Mask	14	R/W	1b	0b – TX source doesn't support Flow14 from table entry 1b – TX source does support Flow14 from table entry
Flow Mask	15	R/W	1b	0b – TX source doesn't support Flow15 from table entry 1b – TX source does support Flow15 from table entry

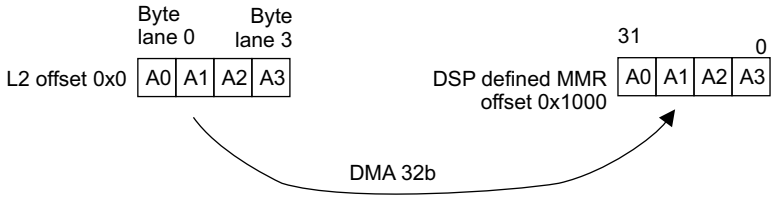
2.3.8 Endianness

RapidIO is based on big endian. This is discussed in detail in section 2.4 of the RapidIO Interconnect specification. Essentially, big endian specifies the address ordering as the most significant bit/byte first. For example, in the 29-bit address field of a RapidIO packet (shown in Figure 6) the left-most bit that is transmitted first in the serial bit stream is the MSB of the address. Likewise, the data payload of the packet is double-word aligned big-endian, which means the MSB is transmitted first. Bit 0 of all the RapidIO-defined MMR registers is the MSB.

All endian specific conversion is handled within the peripheral. For double-word aligned payloads, the data should be written contiguously into memory beginning at the specified address. Any unaligned payloads will be padded and properly aligned within the 8-byte boundary. In this case, WDPTR, RDSIZE, and WRSIZE RapidIO header fields indicate the byte position of the data within the double-word boundary. An example of an unaligned transfer is shown in section 2.4 of the RapidIO Interconnect Specification.

There are no endian translation requirements for accessing the local MMR space. Regardless of the device memory endian configuration, all configuration bus accesses are performed on 32-bit values at a fixed address position. The bit positions in the 32-bit word are defined by this specification. This means that a memory image which will be copied to a MMR is identical between little endian and big endian configurations. Configuration bus reads are performed in the same manner. Figure 29 illustrates the concept. The desired operation is to locally update a serial RapidIO MMR (offset 0x1000) with a value of 0xA0A1A2A3, using the configuration bus.

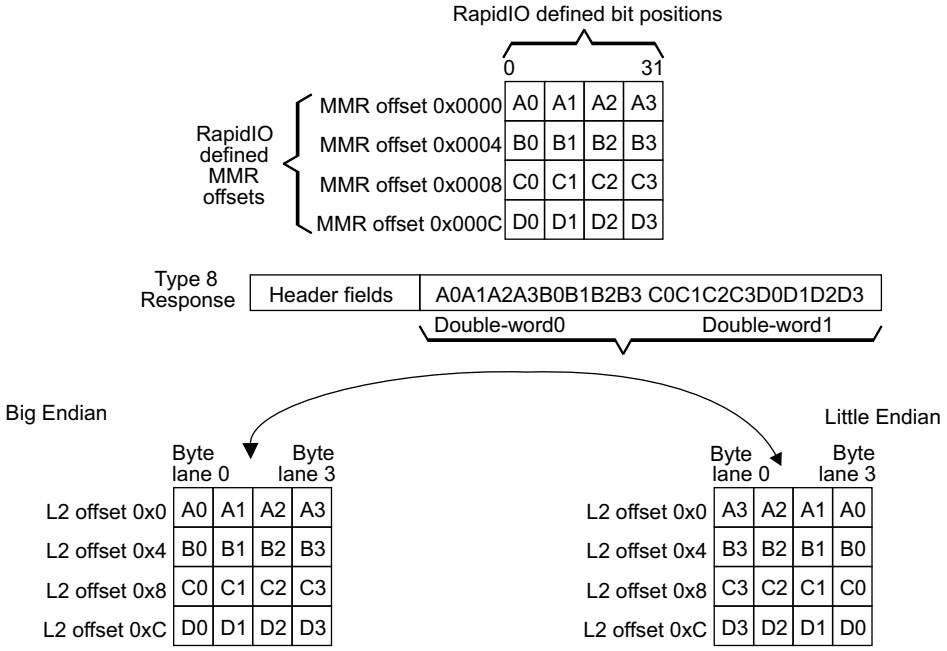
Figure 29. Configuration Bus Example



When accessing RapidIO defined MMR within an external device, RapidIO allows 4B, 8B, or any multiple of a double word access (up to 64B) for Type 8 Maintenance packets. The peripheral only supports 4B accesses as the target, but can generate all sizes of request packets. RapidIO is defined as big endian only, and has double-word aligned big endian packet payloads. The DMA, however, supports byte wide accesses. The peripheral performs endian conversion on the payload if little endian is used on the device. This conversion is not only applicable for Type 8 packets, but is also relevant for all outgoing payloads of NWRITE, NWRITE_R, SWRITE, NREAD, and Message packets. This means that the memory image is different between little endian and big endian configurations, as shown in Figure 30.

Figure 30. DMA Example

DMA Example
The desired operation is to send a Type 8 maintenance request to an external device. The goal is to read 16B of RapidIO MMR from an external device, starting offset 0x0000. This operation involves the LSU block and utilizes the DMA for transferring the response packet payload.



2.3.9 Reset

The RapidIO peripheral allows independent software controlled shutdown for the following blocks: SERDES TX and RX individual ports and PLL, channelized datapath logic (8b/10b, rate handoff FIFO, CRC logic, lane striping/de-skew logic), CPPI module, LSU module, MAU module, and MMR registers. With the exception of BLK_EN0 for the MMR registers, when the BLK*n*_EN signals are deasserted, the clocks are gated to these blocks, effectively providing a shutdown function.

Reset of the SERDES macros is handled independently of the registers discussed in this section. The SERDES can be configured to shutdown unused links or fully shutdown. SERDES TX and RX channels may be enabled/disabled by writing to bit 0 of the SERDES_CFGTX*n*_CNTL and SERDES_CFGRX*n*_CNTL registers. The PLL and remaining SERDES functional blocks can be controlled by writing to the ENPLL signals in the PER_SET_CNTL or SERDES_CFG*n*_CNTL register, depending on device implementation. These registers will drive the SERDES signal inputs, which will gate the reference clock to these blocks internally. This reference clock is sourced from a device pin specifically for the SERDES and is not derived from the CPU clock, thus it resets asynchronously. ENPLL will disable all SERDES high-speed output clocks. Since these clocks are distributed to all the links, ENPLL should only be used to completely shutdown the peripheral. It should be noted that shutdown of SERDES links in between normal packet transmissions is not permissible for two reasons. First, the serial RapidIO sends idle packets between data packets to maintain synchronization and lane alignment. Without this mechanism, the RapidIO RX logic can be mis-aligned for both 1X and 4X ports. Second, the lock time of the SERDES PLL would need to reoccur, which would slow down the operation.

All chip-IO signals must be reset asynchronously to a known state. When the SERDES ENTX signal is held low, the corresponding transmitter is powered down. In this state, both outputs, TXP and TXN, will be pulled high to VDDT.

2.3.9.1 Reset Summary

After reset, the state of the peripheral depends on the default register values and the BLK*n*_EN_INIT tieoff values.

You can also perform a hard reset using the software of each logical block within the peripheral via the GBL_EN and BLK*n*_EN bits. The GBL_EN bits reset the peripheral, while the rest of the device is not reset. The BLK*n*_EN bits shut down unused portions of the peripheral. The *EN functionality will minimize power by resetting the appropriate logical block(s), and gating off the clock to the appropriate logical block(s). This should be considered an abrupt reset independent of the state of the peripheral, and should be capable of resetting the peripheral to its original state.

Upon reset of the peripheral, the device must reestablish communication with its link partner. Depending on the system, this may include a discovery phase in which a host processor reads the peripheral's CAR/CSR registers to determine its capabilities. In its simplest form, it involves retraining the SERDES and going through the initialization phase to synchronize on bit and word boundaries by using idle and control symbols, as described in section 5.5.2 of the Part VI of the RapidIO specification. Until the peripheral and its partner are fully initialized and ready for normal operation, the peripheral will not send any data packets or non-status control symbols.

- GBL_EN_INIT and BLK_EN_INIT[n:0]: These module boundary signals are static tieoffs which control the default state of the GBL_EN and BLK_EN[n:0] MMRs.
- GBL_EN: Resets all MMRs, excluding Reset Ctl Values (0x0000 – 0x1FC). Resets all logical blocks except MMR configuration bus i/f. While asserted, the slave configuration bus is operational.
- BLK_EN0: Resets all MMRs, excluding Reset Ctl Values (0x0000 – 0x1FC). Other logical blocks are unaffected, including MMR configuration bus i/f.
- BLK_EN[n:1]: Single enable/reset per logical block.

2.3.9.2 Enable and Enable Status Registers

Figure 31 through Figure 38 are the register diagrams for the enable and enable status registers.

Figure 31. GBL_EN (Address 0x0030)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 32. GBL_EN_STAT (Address 0x0034)

15	10	9	8
Reserved		BLK8_EN_STAT	BLK7_EN_STAT
R-0		R-1	R-1

7	6	5	4	3	2	1	0
BLK6_EN_STAT	BLK5_EN_STAT	BLK4_EN_STAT	BLK3_EN_STAT	BLK2_EN_STAT	BLK1_EN_STAT	BLK0_EN_STAT	GBL_EN_STAT
R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 33. BLK0_EN (Address 0x0038)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 34. BLK0_EN_STAT (Address 0x003C)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 35. BLK1_EN (Address 0x0040)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 36. BLK1_EN_STAT (Address 0x0044)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, n = value at reset

-
-
-

Figure 37. BLK8_EN (Address 0x0078)

31	1	0
Reserved		EN
R-0		R/W-1

LEGEND: R = Read, W = Write, n = value at reset

Figure 38. BLK8_EN_STAT (Address 0x007C)

31	1	0
Reserved		EN_STAT
R-0		R-1

LEGEND: R = Read, W = Write, n = value at reset

Table 24. Enable and Enable Status Bit Field Descriptions

Name	Bit	Access	Description
GBL_EN	0	R/W	Controls reset to all clock domains within the peripheral. 0 = Peripheral to be disabled (held in reset, clocks disabled) 1 = Peripheral to be enabled
GBL_EN_STAT	0-9	R	Indicates state of GBL_EN reset signal. 0 = Peripheral in reset and all clocks are off 1 = Peripheral enabled and clocking
BLK0_EN	0	R/W	Controls reset to 0th clock/logical domain. By convention, BLK0 should always be the MMR Peripheral control registers. 0 = Logical block 0 to be disabled 1 = Logical block 0 to be enabled
BLK0_EN_STAT	0	R	Indicates state of BLK0_EN reset signal. 0 = Logical block 0 disabled 1 = Logical block 0 enabled

Table 24. Enable and Enable Status Bit Field Descriptions (continued)

Name	Bit	Access	Description
BLK1_EN	0	R/W	Controls reset to logical block 1, which is the LSU. 0 = Logical block 1 disabled (held in reset, clocks disabled) 1 = Logical block 1 enabled
BLK1_EN_STAT	0	R	Indicates state of BLK1_EN reset signal. 0 = Logical block 1 in reset and clock is off 1 = Logical block 1 enabled and clocking
BLK2_EN	0	R/W	Controls reset logical block 2, which is the MAU. 0 = Logical block 2 disabled (held in reset, clocks disabled) 1 = Logical block 2 enabled
BLK2_EN_STAT	0	R	Indicates state of BLK2_EN reset signal. 0 = Logical block 2 reset and clock is off 1 = Logical block 2 enabled and clocking
BLK3_EN	0	R/W	Controls reset logical block 3, which is the TXU. 0 = Logical block 3 disabled (held in reset, clocks disabled) 1 = Logical block 3 enabled
BLK3_EN_STAT	0	R	Indicates state of BLK3_EN reset signal. 0 = Logical block 3 reset and clock is off 1 = Logical block 3 enabled and clocking
BLK4_EN	0	R/W	Controls reset logical block 4, which is the RXU. 0 = Logical block 4 disabled (held in reset, clocks disabled) 1 = Logical block 4 enabled
BLK4_EN_STAT	0	R	Indicates state of BLK4_EN reset signal. 0 = Logical block 4 reset and clock is off 1 = Logical block 4 enabled and clocking
BLK5_EN	0	R/W	Controls reset logical block 5, which is port 0. 0 = Logical block 5 disabled (held in reset, clocks disabled) 1 = Logical block 5 enabled
BLK5_EN_STAT	0	R	Indicates state of BLK5_EN reset signal. 0 = Logical block 5 reset and clock is off 1 = Logical block 5 enabled and clocking
BLK6_EN	0	R/W	Controls reset to logical block 6, which is port 1. 0 = Logical block 6 disabled (held in reset, clocks disabled) 1 = Logical block 6 enabled
BLK6_EN_STAT	0	R	Indicates state of BLK6_EN reset signal. 0 = Logical block 6 in reset and clock is off 1 = Logical block 6 enabled and clocking
BLK7_EN	0	R/W	Controls reset to logical block 7, which is port 2. 0 = Logical block 7 disabled (held in reset, clocks disabled) 1 = Logical block 7 enabled
BLK7_EN_STAT	0	R	Indicates state of BLK7_EN reset signal. 0 = Logical block 7 in reset and clock is off 1 = Logical block 7 enabled and clocking
BLK8_EN	0	R/W	Controls reset to logical block 8, which is port 3. 0 = Logical block 8 disabled (held in reset, clocks disabled) 1 = Logical block 8 enabled

Table 24. Enable and Enable Status Bit Field Descriptions (continued)

Name	Bit	Access	Description
BLK8_EN_STAT	0	R	Indicates state of BLK8_EN reset signal. 0 = Logical block 8 in reset and clock is off 1 = Logical block 8 enabled and clocking

The GBL_EN register is implemented with a single ENABLE bit. This bit is logically ORd with the reset input to the module and is fanned out to all logical blocks within the peripheral.

2.3.9.3 Software Shutdown Details

Power consumption must be minimized for all logical blocks that are in shutdown. In addition to simply asserting the appropriate reset signal to each logical block within the peripheral, clocks are gated off to the corresponding logical block as well. Clocks are allowed to run for 32 clock cycles, which is necessary to fully reset each logical block. When the appropriate logical block is fully reset, the clock input to that subblock is gated off.

When a block is disabled, the reset control block turns off the peripheral with the following sequence:

1. Deassert GBL_EN signal or appropriate BLK*n*_EN signal, which effectively resets all subblocks or the given subblock.
2. Wait 32 clock cycles to guarantee full reset.
3. Gate the input clock to the logical block(s). When the full shutdown procedure is complete, the BLK*n*_EN_STAT bits and/or the GBL_EN_STAT bit contain 0.

The opposite is done for software controlled enabling of a logical block:

1. Assert the BLK_EN signal to release the logical block from reset.
2. Turn on the logical block. When the full start-up procedure is complete, the BLK*n*_EN_STAT bits and/or the GBL_EN_STAT bit contain 1.

When using the GBL_EN to shutdown/reset the peripheral, it is important to first stop all master-initiated commands on the DMA bus interface. For example, if the GBL_EN is asserted in the middle of a DMA transfer from the peripheral, the bus could hang. The procedure is as follows:

1. Stop all RapidIO source transactions, including LSU and TXU operations. This essentially means waiting for the LSU or TXU CC field to be set, or, alternatively, teardown of the active TXU queues.
2. Disable the PEREN bit of the PCR register to stop all new logical layer transactions.
3. Wait the number of clock cycles required to finish any current DMA transfer.
4. Deassert GBL_EN.

2.3.10 Emulation

Expected behavior during emulation halt is controlled within the peripheral by the Peripheral Control register (PCR). This is a single global register that controls emulation for the whole peripheral.

Figure 39. Emulation Control (Peripheral Control Register PCR 0x0004)

31	3	2	1	0
Reserved		PEREN	SOFT	FREE
R-0		R/W-0	R/W-0	R/W-1

LEGEND: R = Read, W = Write, n = value at reset

Table 25. Emulation Control Signals

Name	Bit	Access	Reset Value	Description
Free	0	R/W	1b	FREE = 0, SOFT Bit takes effect FREE = 1, Free run mode (default mode) - Peripheral ignores the EMUSUSP signal and functions normally.
Soft	1	R/W	0b	SOFT = 0 -> Soft Stop (default mode) SOFT = 1 -> Hard stop – All status registers are frozen in default state. (Mode not supported)
PEREN	2	R/W	0b	Peripheral Enable. 0b – Disabled 1b – Enabled
Reserved	31:3		0b	Reserved

Free Run Mode: (default mode) Peripheral does not respond to EMUSUSP assertion. Module functions normally, irrespective of CPU emulation state.

Soft Stop Mode: Peripheral gracefully halts operations. The peripheral halts operation at a point that makes sense both to the internal DMA/data access operation and to the pin interface as described below, after finishing packet reception or transmission in progress:

- **DMA bus DMA master:** DMA bus requests in progress are allowed to complete (DMA bus has no means to throttle command in progress from the Master) DMA bus requests that correspond to the same network packet are allowed to complete. No new DMA bus requests will be generated on the next new packet.
- **Configuration bus MMR interface:** All MMR configuration bus requests are serviced as normal.
- **Events/interrupts:** New events/interrupts are not generated to the CPU for newly arriving packets. Current transactions are allowed to finish and may cause an interrupt upon completion.
- **Slave pin interface:** The pin interface functions as normal. If buffering is available in the peripheral, the peripheral services externally generated requests as long as possible. When the internal buffers are consumed, the peripheral will retry incoming network packets in the physical layer.
- **Master pin interface**: No new master requests are generated. Master requests in progress are allowed to complete, including all packets located in the physical layer transmit buffers.

Hard Stop Mode: Peripheral should halt immediately. This mode is not supported in the RapidIO peripheral.

2.3.11 Initialization Example

2.3.11.1 Enabling the SRIO Peripherals

When the device is powered on, the SRIO peripheral is in a disabled state. Before any SRIO specific initialization can take place, the peripheral needs to be enabled; otherwise, its registers cannot be written, and the reads will all return a value of zero.

```
/* Glb enable srio */
SRIO_REGS->GBL_EN   = 0x00000001 ;
SRIO_REGS->BLK0_EN  = 0x00000001 ;  //MMR_EN
SRIO_REGS->BLK5_EN  = 0x00000001 ;  //PORT0_EN
SRIO_REGS->BLK1_EN  = 0x00000001 ;  //LSU_EN
SRIO_REGS->BLK2_EN  = 0x00000001 ;  //MAU_EN
SRIO_REGS->BLK3_EN  = 0x00000001 ;  //TXU_EN
SRIO_REGS->BLK4_EN  = 0x00000001 ;  //RXU_EN
SRIO_REGS->BLK6_EN  = 0x00000001 ;  //PORT1_EN
SRIO_REGS->BLK7_EN  = 0x00000001 ;  //PORT2_EN
SRIO_REGS->BLK8_EN  = 0x00000001 ;  //PORT3_EN
```

2.3.11.2 PLL, Ports, Device ID and Data Rate Initializations

For example, Enable pll, 333MHz, 4p1x, x20. 3.125 Gbps, full rate, ½ rate, ¼ rate:

```
if (srio4p1x_mode){
       rdata = SRIO_REGS->PER_SET_CNTL;
       wdata = 0x0000014F; 4p1x
       mask  = 0x000001FF;
       mdata = (wdata & mask) | (rdata & ~mask);
       SRIO_REGS->PER_SET_CNTL = mdata ;  // enable pll
}
else{
       wdata =  0x0000004F;  // enable pll, 1p4x
       rdata = SRIO_REGS->PER_SET_CNTL;
       mask  = 0x000001FF;
       mdata = (wdata & mask) | (rdata & ~mask);
       SRIO_REGS->PER_SET_CNTL = mdata ;  // enable pll, 1p1x/4x
}
rdata = SRIO_REGS->SERDES_CFG0_CNTL;
wdata = 0x0000000D;
mask  = 0x00000FFF;
mdata = (wdata & mask) | (rdata & ~mask);
SRIO_REGS->SERDES_CFG0_CNTL    = mdata ;  // JADIS 3.125 Gbps
SRIO_REGS->SERDES_CFG1_CNTL    = mdata ;  // JADIS 3.125 Gbps
SRIO_REGS->SERDES_CFG2_CNTL    = mdata ;  // JADIS 3.125 Gbps
SRIO_REGS->SERDES_CFG4_CNTL    = mdata ;  // JADIS 3.125 Gbps
SRIO_REGS->SERDES_CFGRX0_CNTL  = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX1_CNTL  = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX2_CNTL  = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGRX3_CNTL  = 0x00081101 ;  // enable rx, rate 1
SRIO_REGS->SERDES_CFGTX0_CNTL  = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX1_CNTL  = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX2_CNTL  = 0x00010801 ;  // enable tx, rate 1
SRIO_REGS->SERDES_CFGTX3_CNTL  = 0x00010801 ;  // enable tx, rate 1
```

2.3.11.3 Peripheral Initializations

Set Device ID Registers

```
  rdata = SRIO_REGS->DEVICEID_REG1;
  wdata = 0x00ABBEEF;
  mask  = 0x00FFFFFF;
  mdata = (wdata & mask) | (rdata & ~mask);
  SRIO_REGS->DEVICEID_REG1  = mdata ;  // id-16b=BEEF, id-08b=AB
  rdata = SRIO_REGS->DEVICEID_REG2;
  wdata = 0x00ABBEEF;
mask  = 0x00FFFFFF;
mdata = (wdata & mask) | (rdata & ~mask);
SRIO_REGS->DEVICEID_REG2  = mdata ;  // id-16b=BEEF, id-08b=AB
SRIO_REGS->PER_SET_CNTL      = 0x00000000;    // bootcmpl=0
SRIO_REGS->DEV_ID            = 0xBEEF0030 ;   // id=BEEF, ti=0x0030
SRIO_REGS->DEV_INFO          = 0x00000000 ;   // 0
SRIO_REGS->ASBLY_ID          = 0x00000030 ;   // ti=0x0030
SRIO_REGS->ASBLY_INFO        = 0x00000000;    // 0x0000, next ext=0x0100
SRIO_REGS->PE_FEAT           = 0x20000019 ;   // proc, bu ext, 16-bit ID, 34-bit addr
SRIO_REGS->SW_PORT           = 0x00000400;    // 4 ports
SRIO_REGS->SRC_OP            = 0x0000FDF4;    // all
SRIO_REGS->DEST_OP           = 0x0000FC04;    // all except atomic
SRIO_REGS->PE_LL_CTL         = 0x00000001;    // 34-bit addr
SRIO_REGS->LCL_CFG_HBAR      = 0x00000000 ;   // 0
SRIO_REGS->LCL_CFG_BAR       = 0x00000000;    // 0
SRIO_REGS->BASE_ID           = 0x00ABBEEF;    // 16b-id=BEEF, 08b-id=AB
SRIO_REGS->HOST_BASE_ID_LOCK = 0x0000BEEF;    // id=BEEF, lock
SRIO_REGS->COMP_TAG          = 0x00000000;    // not touched
SRIO_REGS->SP_IP_DISCOVERY_TIMER = 0x90000000;// 0, short cycles for sim
//INIT_MAC0
if (srio4p1x_mode){
          SRIO_REGS->SP_IP_MODE = 0x44000000; // Jadis mltc/rst/pw enable, clear
```

```
}
else{
          SRIO_REGS->SP_IP_MODE = 0x04000000; // Jadis mltc/rst/pw enable, clear
}


SRIO_REGS->IP_PRESCAL        = 0x00000021;  // srv_clk prescalar=0x21 (333MHz)
SRIO_REGS->SP0_SILENCE_TIMER = 0x20000000;  // 0, short cycles for sim
SRIO_REGS->SP1_SILENCE_TIMER = 0x20000000;  // 0, short cycles for sim
SRIO_REGS->SP2_SILENCE_TIMER = 0x20000000;  // 0, short cycles for sim
SRIO_REGS->SP3_SILENCE_TIMER = 0x20000000;  // 0, short cycles for sim
SRIO_REGS->PER_SET_CNTL      = 0x01000000;  // bootcmpl=1
SRIO_REGS->SP_LT_CTL         = 0xFFFFFF00;  // long
SRIO_REGS->SP_RT_CTL         = 0xFFFFFF00;  // long
SRIO_REGS->SP_GEN_CTL        = 0x40000000;  // agent, master, undiscovered
SRIO_REGS->SP0_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP1_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP2_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->SP3_CTL           = 0x00600000;  // enable i/o
SRIO_REGS->ERR_RPT_BH        = 0x00000000;  // next ext=0x0000(last)

SRIO_REGS->ERR_DET           = 0x00000000 ; // clear
SRIO_REGS->ERR_EN            = 0x00000000 ; // disable
SRIO_REGS->H_ADDR_CAPT       = 0x00000000 ; // clear
SRIO_REGS->ADDR_CAPT         = 0x00000000 ; // clear
SRIO_REGS->ID_CAPT           = 0x00000000 ; // clear
SRIO_REGS->CTRL_CAPT         = 0x00000000 ; // clear

SRIO_REGS->SP_IP_MODE        = 0x0000003F;  // mltc/rst/pw enable, clear

SRIO_REGS->SP_IP_PW_IN_CAPT0 = 0x00000000 ; // clear
SRIO_REGS->SP_IP_PW_IN_CAPT1 = 0x00000000 ; // clear
SRIO_REGS->SP_IP_PW_IN_CAPT2 = 0x00000000 ; // clear
SRIO_REGS->SP_IP_PW_IN_CAPT3 = 0x00000000 ; // clear

//INIT_WAIT wait for lane initialization
```

Read register to check portx(1-4) OK bit

```
// polling SRIO_MAC's port_ok bit
rdata = SRIO_REGS->P0_ERR_STAT ;
while ((rdata & 0x00000002) != 0x00000002)
{
         rdata = SRIO_REGS->P0_ERR_STAT ;
}
if (srio4p1x_mode){
         rdata = SRIO_REGS->P1_ERR_STAT;
         while ((rdata & 0x00000002) != 0x00000002)
         {
               rdata = SRIO_REGS->P1_ERR_STAT;
         }
         rdata = SRIO_REGS->P2_ERR_STAT;
         while ((rdata & 0x00000002) != 0x00000002)
         {
               rdata = SRIO_REGS->P2_ERR_STAT;
         }
         rdata = SRIO_REGS->P3_ERR_STAT;
         while ((rdata & 0x00000002) != 0x00000002)
         {
               rdata = SRIO_REGS->P3_ERR_STAT;
         }
}
```

Assert the PEREN bit to enable logical layer data flow

```
SRIO_REGS->PCR = 0x00000004;      // peren
```

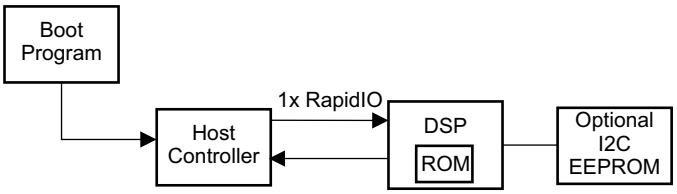
2.3.12 Bootload Capability

2.3.12.1 Configuration

It is assumed that an external device will initiate the bootload data transfer and master the DMA interface. Upon reset, the following sequence of events must occur:

1. DSP is placed in SRIO boot mode by HW mode pins.
2. Host takes DSP out of reset ($\overline{\text{POR}}$ or $\overline{\text{RST}}$). The peripheral's state machines and registers are reset.
3. Internal boot-strap ROM configures device registers, including SERDES, and DMA. DSP executes internal ROM code to initialize SRIO.
   - Choice of 4 pin selectable configurations
   - Optionally, I2C boot can be used to configure SRIO
4. DSP executes idle instruction.
5. RapidIO ports send Idle control symbols to train PHYs.
6. Host enabled to explore system with RapidIO Maintenance packets.
7. Host identifies, enumerates and initializes the RapidIO device.
8. Host controller configures DSP peripherals through maintenance packets.
   - SRIO Device IDs are set for DSPs (either by pin strapping or by host manipulation)
9. Boot Code sent from host controller to DSP L2 memory base address via NWRITE.
10. DSP CPU is awakened by an interrupt such as a RapidIO DOORBELL packet.
11. Boot Code is executed and normal operation follows.

Figure 40. Bootload Operation



2.3.12.2 Bootload Data Movement

The system host is responsible for writing the bootload data into the DSP's L2 memory. As such, bootload is only supported using the Direct I/O model, and not the message passing model. Bootload data must be sent in packets with explicit L2 memory addresses indicating proper destination within the DSP. As part of the peripheral's configuration, it should be set up to transfer the desired bootload program to the DSP's memory through normal DMA bus commands.

2.3.12.3 Device Wakeup

Upon completion of the bootload data transfer, the system host issues a DOORBELL interrupt to the DSP. The RapidIO peripheral processes this interrupt in a manner similar to that described in Section 4, monitoring the DMA bus write-with-response commands to ensure that the data has been completely transferred through the DMA. This interrupt wakes up the CPUs by pulling them out of their reset state.

The 16-bit data field of the DOORBELL packet should be configured to interrupt Core 0 by setting a corresponding ICSR bit as described in Figure 43.

3 Logical/Transport Error Handling and Logging

Error management registers allow detection and logging of logical/transport layer errors. Figure 41 illustrates the detectable errors.

Figure 41. Detectable Errors

31	30	29	28	27	26	25
IO ERR Rspns	Msg ERR Rspns	GSM ERR Rspns	ERR MSG Format	ILL Trans Decode	ILL Trans Trgt ERR	MSG REQ Timeout
R/W0c, 0b	R/W0c, 0b	R, 0b	R/W0c, 0b	R/W0c, 0b	R, 0b	R/W0c, 0b
24	23	22	21	7	6	5
PKT Rspns Timeout	Unsolicited Rspns	Unsupported Trans	Rsv	RX_CPPI Security	RX IO Access	Rsv
R/W0c, 0b	R/W0c, 0b	R/W0c, 0b	R, All 0s	R/W0c, 0b	R/W0c, 0b	R, All 0s

The peripheral supports all detectable errors, except bits 29 and 26. The functional blocks involved for each detectable error condition are listed below, along with a brief description of the capture register contents.

Logical Layer Error Detect CSR:

bit 31: LSU (request packet is logged)
bit 30: TXU (request packet is logged)
bit 29: Not supported
bit 28: RXU (request packet is logged)
bit 27: LSU and TXU (response packet is logged), MAU and RXU (request packet is logged)
bit 26: Not supported
bit 25: RXU (request packet is logged)
bit 24: LSU and TXU (request packet is logged)
bit 23: LSU and TXU (response packet is logged
bit 22: MAU (request packet is logged)
...
bit 7: RXU (request packet is logged)
bit 6: MAU (request packet is logged)

4 Interrupt Conditions

This section defines the CPU interrupt capabilities and requirements of the peripheral.

4.1 CPU Interrupts

The following interrupts are supported by the RIO peripheral.

- Error Status: Event indicating that a run-time error was reached. The CPU should reset/resynchronize the peripheral.
- Critical Error: Event indicating that a critical error state was reached. The CPU should reset the system.
- CPU servicing: Event indicating that the CPU should service the peripheral.

4.2 General Description

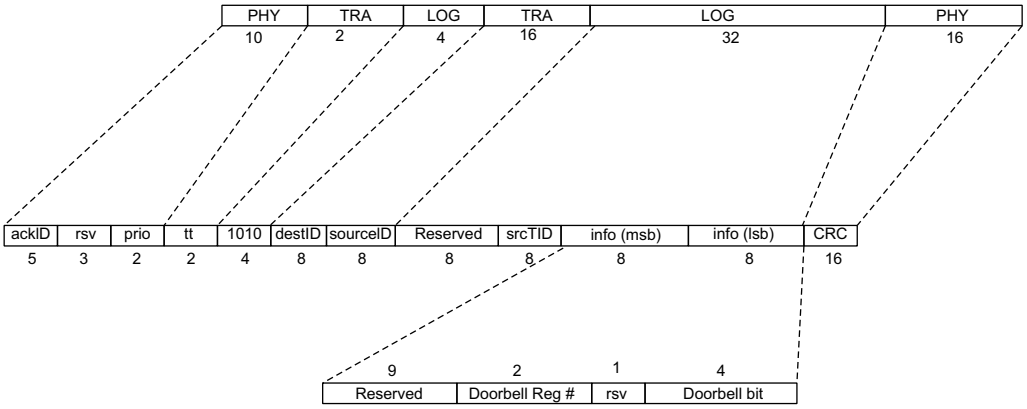
The RIO peripheral is capable of generating various types of CPU interrupts. The interrupts serve two general purposes: error indication and servicing requests.

Since RapidIO is a packet oriented interface, the peripheral must recognize and respond to inbound signals from the serial interface. There are no GPIO or external pins used to indicate an interrupt request. Thus, the interrupt requests are signaled either by an external RapidIO device through the packet protocols discussed as follows, or are generated internally by the RIO peripheral.

CPU servicing interrupts lag behind the corresponding data, which was generally transferred from an external processing element into local L2 memory. This transfer can use a messaging or direct I/O protocol. When the single or multi-packet data transfer is complete, the external PE, or the peripheral itself, must notify the local processor that the data is available for processing. To avoid erroneous data being processed by the local CPU, the data transfer must complete through the DMA before the CPU interrupt is serviced. This condition could occur since the data and interrupt queues are independent of each other, and DMA transfers can stall. To avoid this condition, all data transfers from the peripheral through the DMA use write-with-response DMA bus commands, allowing the peripheral to always be aware that outstanding transfers have completed. Interrupts are generated only after all DMA bus responses are received. Since all RapidIO packets are handled sequentially, and submitted on the same DMA priority queue, the peripheral must keep track of the number of DMA requests submitted and the number of responses received. Thus, a simple counter within the peripheral ensures that data packets have arrived in memory before submitting an interrupt.

The sending device initiates the interrupt by using the RapidIO defined DOORBELL message. The DOORBELL packet format is shown in Figure 42. The DOORBELL functionality is user-defined. This packet type is commonly used to initiate CPU interrupts. A DOORBELL packet is not associated with a particular data packet that was previously transferred, so the INFO field of the packet must be configured to reflect the DOORBELL bit to be serviced for the correct TID info to be processed.

Figure 42. RapidIO DOORBELL Packet for Interrupt Use



The DOORBELL packet's 16-bit INFO field indicates which DOORBELL register interrupt bit to set. There are four DOORBELL registers, each currently with 16 bits, allowing 64 interrupt sources or circular buffers. Each bit can be assigned to any core as described by the Interrupt Condition Routing Registers. Additionally, each status bit is user-defined for the application. For instance, it may be desirable to support multiple priorities with multiple TID circular buffers per core if control data uses a high priority (i.e., priority = 2), while data packets are sent on priority 0 or 1. This allows the control packets to have preference in the switch fabric and arrive as quickly as possible. Since it may be required to interrupt the CPU for both data and control packet processing separately, separate circular buffers are used, and DOORBELL packets must distinguish between them for interrupt servicing. If any reserved bit in the DOORBELL info field is set, an error response is sent.

The interrupt approach to the messaging protocol is somewhat different. Since the source device is unaware of the data's physical location in the destination device, and since each messaging packet contains size and segment information, the peripheral can automatically generate the interrupt after it has successfully received all packet segments comprising the complete message. This DMA interface uses the Communications Port Programming Interface (CPPI). This interface is a link-listed approach versus a circular buffer approach. Data buffer descriptors which contain information such as start of Packet (SOP), end of packet (EOP), end of queue (EOQ), and packet length are built from the RapidIO header fields. The data buffer descriptors also contain the address of the corresponding data buffer as assigned by the receive device. The data buffer descriptors are then link-listed together as multiple packets are received. Interrupts are generated by the peripheral after all segments of the messages are received and successfully transferred through the DMA bus with the write-with-response commands. Interrupt pacing is also implemented at the peripheral level to manage the interrupt rate, as described in Section 4.6.

Error handling on the RapidIO link is handled by the peripheral, and as such, does not require the intervention of software for recovery. This includes CRC errors due to bit rate errors that may cause erroneous or invalid operations. The exception to this statement is the use of the RapidIO error management extended features. This specification monitors and tabulates the errors that occur on a per port basis. If the number of errors exceeds a pre-determined configurable amount, the peripheral should interrupt the CPU software and notify that an error condition exits. Alternatively, if a system host is used, the peripheral may issue a port-write operation to notify the system software of a bad link.

A system reset, or Critical Error interrupt, can be initialized through the RapidIO link. This procedure allows an external device to reset the local device, causing all state machine and configuration registers to reset to their original values. This is executed with the Reset-Device command described in Part VI, Section 3.4.5 of the Serial specification. Four sequential Reset-Device control symbols are needed to avoid inadvertent resetting of a device.

4.3 Interrupt Condition Control Registers

Interrupt condition control registers configure which CPU interrupts are to be generated and how, based on the peripheral activity. All peripheral conditions that result in a CPU interrupt are grouped so that the interrupt can be accessed in the minimum number of register reads possible.

For each of the three types of interrupts (CPU servicing, error status, and critical error), there are two sets of registers:

- Interrupt Condition Status Register (ICSR): Status register that reflects the state of each condition that can trigger the interrupt.
- Interrupt Condition Clear Register (ICCR): Command register that allows each condition to be cleared. This is typically required prior to enabling a condition, such that spurious interrupts are not generated.

These registers are accessible in the memory map of the CPU. The CPU controls the clear register. The status register is readable by the CPU to determine the peripheral condition.

Table 26. Interrupt Source Configuration Options

Field	Access	Reset Value	Value	Function
ICSx	R	0	0b	Condition not present
			1b	Condition present
ICCx	W	0	0b	No effect
			1b	Condition status cleared

Figure 43. DOORBELL0 Interrupt Registers for Direct I/O Transfers

DOORBELL0 Interrupt Condition Status Registers (ICSR) (Address Offset 0x0200)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

DOORBELL0 Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0208)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - Doorbell0, bit 0 through ICS15 - Doorbell0, bit 15.

Figure 44. DOORBELL1 Interrupt Registers for Direct I/O Transfers

DOORBELL1 Interrupt Condition Status Registers (ICSR) (Address Offset 0x0210)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

DOORBELL1 Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0218)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - Doorbell1, bit 0, through ICS15 - Doorbell1, bit 15.

Figure 45. DOORBELL2 Interrupt Registers for Direct I/O Transfers

DOORBELL2 Interrupt Condition Status Registers (ICSR) (Address Offset 0x0220)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

DOORBELL2 Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0228)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - Doorbell2, bit 0, through ICS15 - Doorbell2, bit 15.

Figure 46. DOORBELL3 Interrupt Registers for Direct I/O Transfers

DOORBELL3 Interrupt Condition Status Registers (ICSR) (Address Offset 0x0230)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

DOORBELL3 Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0238)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - Doorbell3, bit 0, through ICS15 - Doorbell3, bit 15.

Figure 47. RX_CPPI Interrupts Using Messaging Mode Data Transfers

RX_CPPI Interrupt Condition Status Registers (ICSR) (Address Offset 0x0240)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

RX_CPPI Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0248)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - RX CPPI interrupt, buffer descriptor queue 0, through ICS15 - RX CPPI interrupt, buffer descriptor queue 15.

Clearing of any ICSR bit depends on the CPU writing the value of last buffer descriptor processed to the RX DMA State CP. Port hardware clears the ICSR bit only if the CP value written by the CPU equals the port written value in the RX DMA State CP register.

Figure 48. TX _CPPI Interrupts Using Messaging Mode Data Transfers

TX_CPPI Interrupt Condition Status Registers (ICSR) (Address Offset 0x0250)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

TX_CPPI Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0258)

31															16
Reserved															
R-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where ICS0 - TX CPPI interrupt, buffer descriptor queue 0, through ICS15 - TX CPPI interrupt, buffer descriptor queue 15.

Clearing of any ICSR bit is dependent on the CPU writing to the TX DMA State CP. The CPU acknowledges the interrupt after reclaiming all available buffer descriptors by writing the CP value. This value is compared against the port written value in the TX DMA State CP register. If equal, the interrupt is deasserted.

Figure 49. LSU Load/Store Module Interrupts

LSU Interrupt Condition Status Registers (ICSR) (Address Offset 0x0260)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICS31	ICS30	ICS29	ICS28	ICS27	ICS26	ICS25	ICS24	ICS23	ICS22	ICS21	ICS20	ICS19	ICS18	ICS17	ICS16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICS15	ICS14	ICS13	ICS12	ICS11	ICS10	ICS9	ICS8	ICS7	ICS6	ICS5	ICS4	ICS3	ICS2	ICS1	ICS0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

LSU Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0268)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ICC31	ICC30	ICC29	ICC28	ICC27	ICC26	ICC25	ICC24	ICC23	ICC22	ICC21	ICC20	ICC19	ICC18	ICC17	ICC16
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICC15	ICC14	ICC13	ICC12	ICC11	ICC10	ICC9	ICC8	ICC7	ICC6	ICC5	ICC4	ICC3	ICC2	ICC1	ICC0
W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where:

- Bit 0- Transaction complete, No Errors (Posted/Non-posted), LSU1 – see note
- Bit 1- Non-posted transaction received ERROR response, or error in response payload, LSU1
- Bit 2- Transaction was not sent due to Xoff condition, LSU1
- Bit 3- Transaction was not sent due to unsupported transaction type or invalid field encoding LSU1
- Bit 4- Transaction Timeout Occurred, LSU1
- Bit 5- Transaction was not sent due to DMA data transfer error, LSU1
- Bit 6- Retry Doorbell response received or Atomic Test-and-swap was not allowed (semaphore in use), LSU1
- Bit 7- Packet not sent due to unavailable outbound credit at given priority, LSU1
- Bit 8- Transaction complete, No Errors (Posted/Non-posted), LSU2 – see note
- Bit 9- Non-posted transaction received ERROR response, or error in response payload, LSU2
- Bit 10- Transaction was not sent due to Xoff condition, LSU2
- Bit 11- Transaction was not sent due to unsupported transaction type or invalid field encoding LSU2
- Bit 12- Transaction Timeout Occurred, LSU2
- Bit 13- Transaction was not sent due to DMA data transfer error, LSU2
- Bit 14- Retry Doorbell response received or Atomic Test-and-swap was not allowed (semaphore in use), LSU2
- Bit 15- Packet not sent due to unavailable outbound credit at given priority, LSU2
- Bit 16- Transaction complete, No Errors (Posted/Non-posted), LSU3 – see note
- Bit 17- Non-posted transaction received ERROR response, or error in response payload, LSU3
- Bit 18- Transaction was not sent due to Xoff condition, LSU3
- Bit 19- Transaction was not sent due to unsupported transaction type or invalid field encoding LSU3
- Bit 20- Transaction Timeout Occurred, LSU3

- Bit 21- Transaction was not sent due to DMA data transfer error, LSU3
- Bit 22- Retry Doorbell response received or Atomic Test-and-swap was not allowed (semaphore in use), LSU3
- Bit 23- Packet not sent due to unavailable outbound credit at given priority, LSU3
- Bit 24- Transaction complete, No Errors (Posted/Non-posted), LSU4 – see note
- Bit 25- Non-posted transaction received ERROR response, or error in response payload, LSU4
- Bit 26- Transaction was not sent due to Xoff condition, LSU4
- Bit 27- Transaction was not sent due to unsupported transaction type or invalid field encoding LSU4
- Bit 28- Transaction Timeout Occurred, LSU4
- Bit 29- Transaction was not sent due to DMA data transfer error, LSU4
- Bit 30- Retry Doorbell response received or Atomic Test-and-swap was not allowed (semaphore in use), LSU4
- Bit 31- Packet not sent due to unavailable outbound credit at given priority, LSU4

Note: Enable for this interrupt is ultimately controlled by the Interrupt Req register bit of the Load/Store command registers. This allows enabling/disabling on a per request basis. For optimum LSU performance, interrupt pacing should not be used on the LSU interrupts. Section 4.6 describes interrupt pacing.

Figure 50. ERR_RST_EVNT Error, Reset, and Special Event Interrupt

ERR_RST_EVNT Interrupt Condition Status Registers (ICSR) (Address Offset 0x0270)

31															17	16
Reserved																ICS16
R-0																R/W-0

15				12	11	10	9	8	7			3	2	1	0
Reserved					ICS11	ICS10	ICS9	ICS8	Reserved				ICS2	ICS1	ICS0
R-0					R/W-0	R/W-0	R/W-0	R/W-0	R-0				R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

ERR_RST_EVNT Interrupt Condition Clear Registers (ICCR) (Address Offset 0x0278)

31															17	16
Reserved																ICC16
R-0																W-0

15				12	11	10	9	8	7			3	2	1	0
Reserved					ICC11	ICC10	ICC9	ICC8	Reserved				ICC2	ICC1	ICC0
R-0					W-0	W-0	W-0	W-0	R-0				W-0	W-0	W-0

LEGEND: R = Read, W = Write, n = value at reset

Where:

- Bit 0 - Multi-cast event control symbol interrupt received on any port
- Bit 1 - Port-write-In request received on any port
- Bit 2 - Logical Layer Error Management Event Capture
- Bit 8 - Port0 Error
- Bit 9 - Port1 Error
- Bit 10 - Port2 Error
- Bit 11 - Port3 Error
- Bit 16 - Device Reset Interrupt from any port

The interrupt conditions are programmable to select the interrupt output that will be driven. Each condition is independently programmable to use any of the interrupt destinations supported by the device. For example, a quad core device may support four CPU servicing interrupt destinations, one per core (INTDST0 for Core0, INTDST1 for Core1, INTDST2 for Core2, and INTDST3 for Core3). In addition, INTDST4 may be globally routed to all cores and provide notification of a change in the Error Status interrupt ICSR. INTDST5 may be globally routed to all cores and provide notification of a change in the Device Reset interrupt ICSR. The routing defaults are shown below.

Table 27. Interrupt Condition Routing Options

Field	Access	Reset Value	Value	Function
ICRx	R	0000b	0000b	Routed to INTDST0
			0001b	Routed to INTDST1
			0010b	Routed to INTDST2
			0011b	Routed to INTDST3
			0100b	Routed to INTDST4
			0101b	Routed to INTDST5
			0110b	Routed to INTDST6
			0111b	Routed to INTDST7
			1111b	No interrupt destination, interrupt source disabled
			other	Reserved

Figure 51. Doorbell 0 Interrupt Condition Routing Registers

DOORBELL0_ICRR (Address Offset 0x280)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000
15–12	**11–8**	**7–4**	**3–0**
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

DOORBELL0_ICRR2 (Address Offset 0x284)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000
15–12	**11–8**	**7–4**	**3–0**
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Other ICRRs have the same bit field map, with the following addresses:

- DOORBELL1_ICRR and DOORBELL1_ICCR2 (Address Offset 0x0290 and 0x0294)
- DOORBELL2_ICRR and DOORBELL2_ICRR2 (Address Offset 0x02A0 and 0x02A4)
- DOORBELL3_ICRR and DOORBELL3_ICRR2 (Address Offset 0x02B0 and 0x02B4)
- RX CPPI_ICRR and RX CPPI_ICRR2 (Address Offset 0x02C0 and 0x02C4)
- TX CPPI_ICRR and TX CPPI_ICRR2 (Address Offset 0x02D0 and 0x02D4)

Figure 52. Load/Store Module Interrupt Condition Routing Registers

LSU_ICRR0 (Address Offset 0x02E0)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

LSU_ICRR1 (Address Offset 0x02E4)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

LSU_ICRR2 (Address Offset 0x02E8)

31–28	27–24	23–20	19–16
ICR23	ICR22	ICR21	ICR20
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR19	ICR18	ICR17	ICR16
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

LSU_ICRR3 (Address Offset 0x02EC)

31–28	27–24	23–20	19–16
ICR31	ICR30	ICR29	ICR28
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR27	ICR26	ICR25	ICR24
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Figure 53. Error, Reset, and Special Event Interrupt Condition Routing Registers

ERR_RST_EVNT_ICRR (Address Offset 0x02F0)

31 12	11 8	7 4	3 0
Reserved	ICR2	ICR1	ICR0
R-0	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

ERR_RST_EVNT_ICRR2 (Address Offset 0x02F4)

31 16
Reserved
R-0

15 12	11 8	7 4	3 0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

ERR_RST_EVNT_ICRR3 (Address Offset 0x02F8)

31 4	3 0
Reserved	ICR16
R-0	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

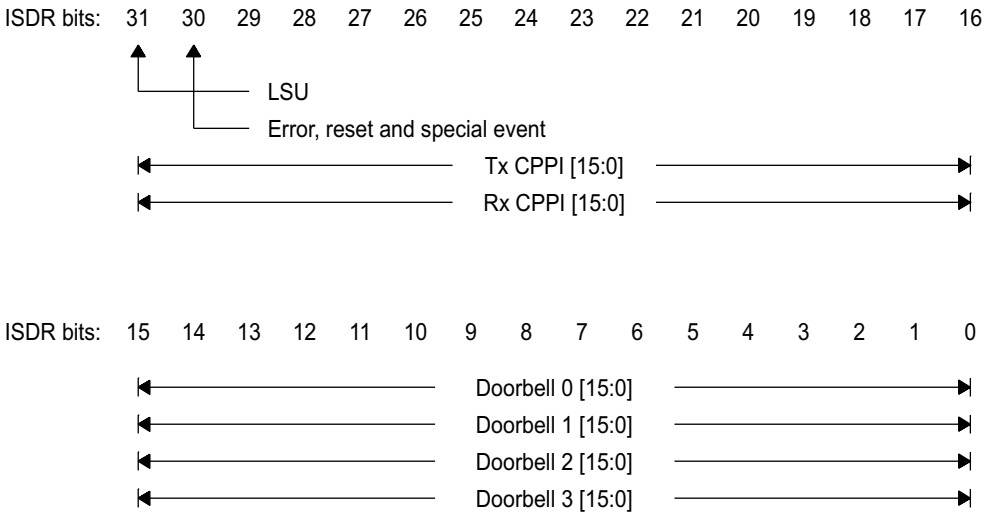
4.4 Interrupt Status Decode Registers

There are 8 blocks of the ICSRs to indicate the source of a pending interrupt.

```
0x0200: Doorbell0 interrupts
0x0210: Doorbell1 interrupts
0x0220: Doorbell2 interrupts
0x0230: Doorbell3 interrupts
0x0240: Rx CPPI interrupts
0x0250: Tx CPPI interrupts
0x0260: LSU interrupts
0x0270: Error, Reset, and Special Event interrupts
```

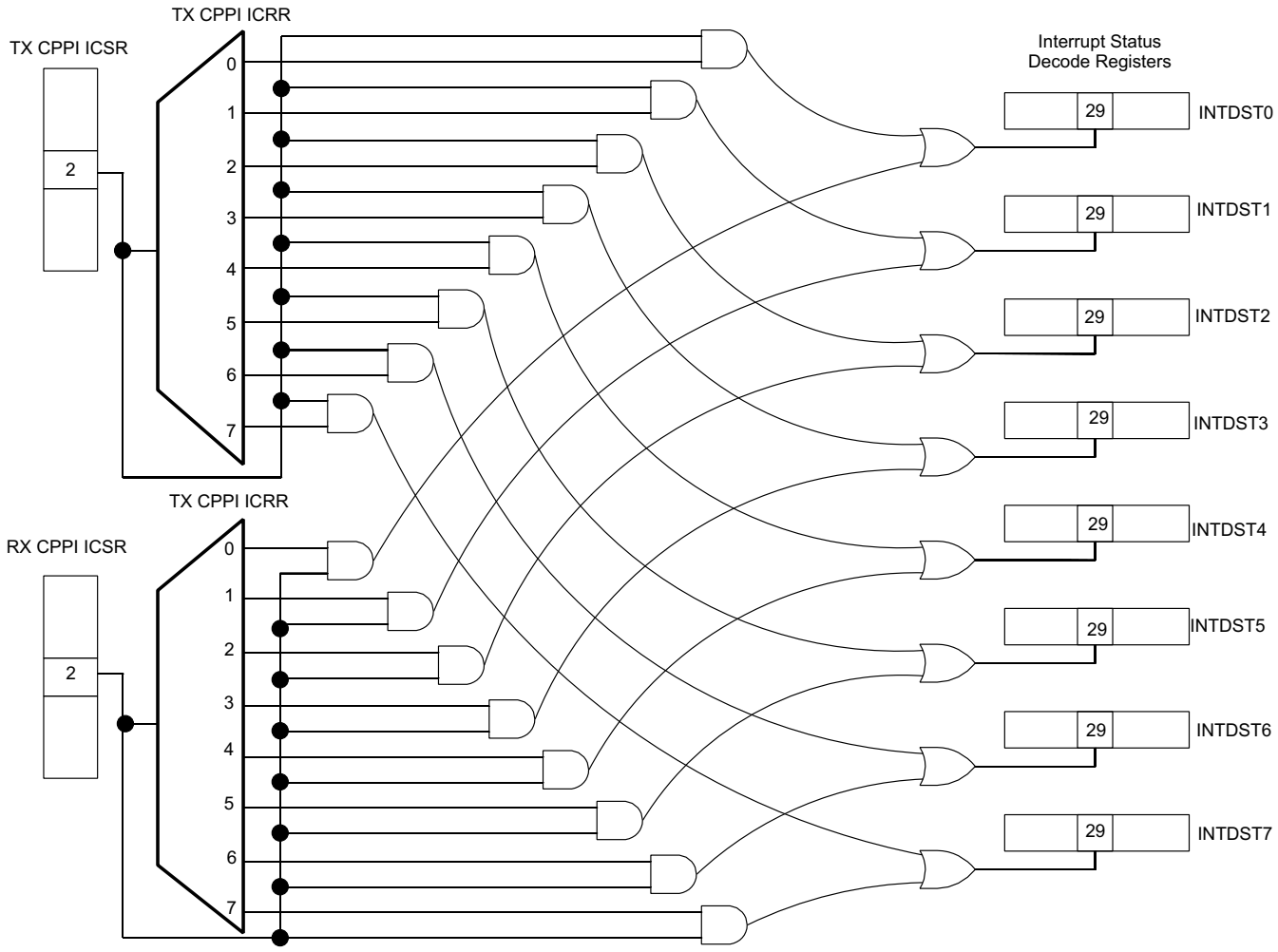
To reduce the number of reads (up to 5 reads) required to find the source bit, an Interrupt Status Decode Register (ISDR) is implemented for each supported physical interrupt (INTDST0 – INTDST7). These registers are shown in Figure 56. Interrupt sources that select a particular physical interrupt destination are mapped to specific bits in the decode register. The interrupt sources are only mapped to an interrupt decode register if the ICRR routes the interrupt source to the corresponding physical interrupt. The status decode bit is a logical OR of multiple interrupt sources that are mapped to the same bit. The mapping of interrupt source bits to decode bits is fixed and non-programmable, as follows:

Figure 54. Sharing of ISDR Bits



As an example, if bit 29 of the ISDR is set, this indicates that there is a pending interrupt on either the TX CPPI queue 2 or RX CPPI queue 2. Figure 55 illustrates the decode routing.

Figure 55. Example Diagram of Interrupt Status Decode Register Mapping



LSU bits within the ICSR are logically grouped for a given core and ORd together into a single bit of the decode register. Similarly, the Error/Reset/Special event bits within the ICSR are ORd together into a single bit of the decode register. When either of these bits are set in the decode register, the CPU must make additional reads to the corresponding ICSRs to determine that exact interrupt source. It is recommended to arrange the Doorbell ICSRs per core, so that the CPU can determine the Doorbell interrupt source from a single read of the decode register. If the RX and TX CPPI queues are assigned orthogonally to different cores, the CPU can determine the CPPI interrupt source from a single read of the decode registers as well. The only exception to this is bit 31 and 30 of the decode register (TX and RX CPPI Queues 19 and 18) which are ORd with the LSU and Error bit, respectively.

Figure 56. INTDST*n*_Decode Interrupt Status Decode Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ISDR31	ISDR30	ISDR29	ISDR28	ISDR27	ISDR26	ISDR25	ISDR24	ISDR23	ISDR22	ISDR21	ISDR20	ISDR19	ISDR18	ISDR17	ISDR16
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ISDR15	ISDR14	ISDR13	ISDR12	ISDR11	ISDR10	ISDR9	ISDR8	ISDR7	ISDR6	ISDR5	ISDR4	ISDR3	ISDR2	ISDR1	ISDR0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

LEGEND: R = Read, W = Write, n = value at reset

Offsets:

- INTDST0 – 0x0300
- INTDST1 – 0x0304
- INTDST2 – 0x0308
- INTDST3 – 0x030C
- INTDST4 – 0x0310
- INTDST5 – 0x0314
- INTDST6 – 0x0318
- INTDST7 – 0x031C

4.5 Interrupt Generation

Interrupts are triggered on a 0-to-1 logic-signal transition. Regardless of the interrupt sources, the physical interrupts are set only when the total number of set ICSR bits transitions from none to one or more. The peripheral is responsible for setting the correct bit within the ICSR. The ICRR register maps the pending interrupt request to the appropriate physical interrupt line. The corresponding CPU is interrupted and reads the ISDR and ICSR registers to determine the interrupt source and appropriate action. Interrupt generation is governed by the interrupt pacing discussed Section 4.6.

4.6 Interrupt Pacing

The rate at which an interrupt can be generated is controllable for each physical interrupt destination. Rate control is implemented with a programmable down-counter. The load value of the counter is written by the CPU into the appropriate interrupt rate control register (see Figure 57). The counter reloads and immediately starts down-counting each time the CPU writes these registers. When the rate control counter register is written, and the counter value reaches zero (note that the CPU may write zero immediately for a zero count), the interrupt pulse generation logic is allowed to fire a single pulse if any bits in the corresponding ICSR register bits are set (or become set after the zero count is reached). The counter remains at zero. When the single pulse is generated, the logic will not generate another pulse, regardless of interrupt status changes, until the rate control counter register is written again. If interrupt pacing is not desired for a particular INTDST, the CPU must still write 0x00000000 into the INTDST*n*_RATE_CNTL register after clearing the corresponding ICSR bits to acknowledge the physical interrupt. If an ICSR is not mapped to an interrupt destination, pending interrupt bits within the ICSR maintain current status. Once enabled, the interrupt logic re-evaluates all pending interrupts and re-pulses the interrupt signal if any interrupt conditions are pending. The down-counter is based on the DMA clock cycle.

Figure 57. INTDST*n*_RATE_CNTL Interrupt Rate Control Register

31	0
32-bit Count Down Value	
R/W-0	

LEGEND: R = Read, W = Write, n = value at reset

Offsets:

- INTDST0 – 0x0320
- INTDST1 – 0x0324
- INTDST2 – 0x0328
- INTDST3 – 0x032C
- INTDST4 – 0x0330
- INTDST5 – 0x0334
- INTDST6 – 0x0338
- INTDST7 – 0x033C

4.7 Interrupt Handling

Interrupts are either signaled externally through RapidIO packets, or internally by state machines in the peripheral. CPU servicing interrupts are signaled externally by the DOORBELL RapidIO packet in Direct I/O mode, or internally by the CPPI module (described in section 8) in the message passing mode. Error Status interrupts are signaled when error counting logic within the peripheral have reached their thresholds. In either case, it is the peripheral that signals the interrupt and sets the corresponding status bits.

When the CPU is interrupted, it reads the ICSR registers to determine the source of the interrupt and appropriate action to take. For example, if it is a DOORBELL interrupt, the CPU will read from an L2 address that is specified by its circular buffer read pointer that is managed by software. There may be more than one circular buffer for each core. The correct circular buffer to read from and increment depends on the bit set in the ICSR register. The CPU then clears the status bit.

For Error Status interrupts, the peripheral must indicate to all the CPUs that one of the link ports has reached the error threshold. In this case, the peripheral sets the status bit indicating degraded or failed limits have been reached, and an interrupt is generated to each core through the ICRR mapping. The cores can then scan the ICSR registers to determine the port with the error problems. Further action can then be taken as determined by the application.

```
Interrupt Handler
temp1 = SRIO_REGS->TX_CPPI_ICSR;
  if ((temp1 & 0x00000001) == 0x00000001)
  {
           SRIO_REGS->Queue0_TxDMA_CP = (int )TX_DESCP0_0;
  }

temp2 = SRIO_REGS->RX_CPPI_ICSR;
  if ((temp2 & 0x00000001) == 0x00000001)
  {
           SRIO_REGS->Queue0_RXDMA_CP = (int )RX_DESCP0_0;
  }

  interruptStatus[0]  = SRIO_REGS->DOORBELL3_ICSR;
  interruptStatus[1]  = SRIO_REGS->DOORBELL3_ICCR;
  interruptStatus[2]  = SRIO_REGS->LSU_ICSR;
  interruptStatus[3]  = SRIO_REGS->LSU_ICCR;
  interruptStatus[4]  = SRIO_REGS->DOORBELL3_ICRR;
  interruptStatus[5]  = SRIO_REGS->DOORBELL3_ICRR2;
  interruptStatus[6]  = SRIO_REGS->LSU_ICRR;
  interruptStatus[7]  = SRIO_REGS->LSU_ICRR2;
  interruptStatus[8]  = SRIO_REGS->INTDST0_Decode;
  interruptStatus[9]  = SRIO_REGS->ERR_RST_EVNT_ICRR;
  interruptStatus[10] = SRIO_REGS->INTDST0_Rate_CNTL;
```

```
interruptStatus[11] = SRIO_REGS->ERR_RST_EVNT_ICSR;
interruptStatus[12] = SRIO_REGS->ERR_RST_EVNT_ICCR;

SRIO_REGS->DOORBELL0_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL1_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL2_ICCR=0xFFFFFFFF;
SRIO_REGS->DOORBELL3_ICCR=0xFFFFFFFF;
SRIO_REGS->INTDST0_Rate_CNTL=1;

SRIO_REGS->LSU_ICCR = 0xFFFF;
SRIO_REGS->INTDST1_Rate_CNTL = 1;
```

5 SRIO Registers

5.1 Introduction

Table 28 lists the memory-mapped registers for the Serial Rapid IO (SRIO). See the device-specific data manual for the memory address of these registers.

Table 28. Serial Rapid IO (SRIO) Registers

Offset	Acronym	Register Description	Section
0x0000	PID	Peripheral Identification Register	Section 5.2
0x0004	PCR	Peripheral Control Register	Section 5.3
0x0020	PER_SET_CNTL	Peripheral Settings Control Register	Section 5.4
0x0030	GBL_EN	Peripheral Global Enable Register	Section 5.5
0x0034	GBL_EN_STAT	Peripheral Global Enable Status	Section 5.6
0x0038	BLK0_EN	Block Enable 0	Section 5.7
0x003C	BLK0_EN_STAT	Block Enable Status 0	Section 5.8
0x0040	BLK1_EN	Block Enable 1	Section 5.7
0x0044	BLK1_EN_STAT	Block Enable Status 1	Section 5.8
0x0048	BLK2_EN	Block Enable 2	Section 5.7
0x004C	BLK2_EN_STAT	Block Enable Status 2	Section 5.8
0x0050	BLK3_EN	Block Enable 3	Section 5.7
0x0054	BLK3_EN_STAT	Block Enable Status 3	Section 5.8
0x0058	BLK4_EN	Block Enable 4	Section 5.7
0x005C	BLK4_EN_STAT	Block Enable Status 4	Section 5.8
0x0060	BLK5_EN	Block Enable 5	Section 5.7
0x0064	BLK5_EN_STAT	Block Enable Status 5	Section 5.8
0x0068	BLK6_EN	Block Enable 6	Section 5.7
0x006C	BLK6_EN_STAT	Block Enable Status 6	Section 5.8
0x0070	BLK7_EN	Block Enable 7	Section 5.7
0x0074	BLK7_EN_STAT	Block Enable Status 7	Section 5.8
0x0078	BLK8_EN	Block Enable 8	Section 5.7
0x007C	BLK8_EN_STAT	Block Enable Status 8	Section 5.8
0x0080	DEVICEID_REG1	RapidIO DEVICEID1 Register	Section 5.9
0x0084	DEVICEID_REG2	RapidIO DEVICEID2 Register	Section 5.10
0x0090	PF_16B_CNTL0	Packet Forwarding Register 0 for 16b DeviceIDs	Section 5.11
0x0094	PF_8B_CNTL0	Packet Forwarding Register 0 for 8b DeviceIDs	Section 5.12
0x0098	PF_16B_CNTL1	Packet Forwarding Register 1 for 16b DeviceIDs	Section 5.11
0x009C	PF_8B_CNTL1	Packet Forwarding Register 1 for 8b DeviceIDs	Section 5.12
0x00A0	PF_16B_CNTL2	Packet Forwarding Register 2 for 16b DeviceIDs	Section 5.11
0x00A4	PF_8B_CNTL2	Packet Forwarding Register 2 for 8b DeviceIDs	Section 5.12
0x00A8	PF_16B_CNTL3	Packet Forwarding Register 3 for 16b DeviceIDs	Section 5.11
0x00AC	PF_8B_CNTL3	Packet Forwarding Register 3 for 8b DeviceIDs	Section 5.12
0x0100	SERDES_CFGRX0_CNTL	SERDES Receive Channel Configuration Register 0	Section 5.13
0x0104	SERDES_CFGRX1_CNTL	SERDES Receive Channel Configuration Register 1	Section 5.13
0x0108	SERDES_CFGRX2_CNTL	SERDES Receive Channel Configuration Register 2	Section 5.13
0x010C	SERDES_CFGRX3_CNTL	SERDES Receive Channel Configuration Register 3	Section 5.13

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x0110	SERDES_CFGTX0_CNTL	SERDES Transmit Channel Configuration Register 0	Section 5.14
0x0114	SERDES_CFGTX1_CNTL	SERDES Transmit Channel Configuration Register 1	Section 5.14
0x0118	SERDES_CFGTX2_CNTL	SERDES Transmit Channel Configuration Register 2	Section 5.14
0x011C	SERDES_CFGTX3_CNTL	SERDES Transmit Channel Configuration Register 3	Section 5.14
0x0120	SERDES_CFG0_CNTL	SERDES Macro Configuration Register 0	Section 5.15
0x0124	SERDES_CFG1_CNTL	SERDES Macro Configuration Register 1	Section 5.15
0x0128	SERDES_CFG2_CNTL	SERDES Macro Configuration Register 2	Section 5.15
0x012C	SERDES_CFG3_CNTL	SERDES Macro Configuration Register 3	Section 5.15
0x0200	DOORBELL0_ICSR	DOORBELL Interrupt Status Register 0	Section 5.16
0x0208	DOORBELL0_ICCR	DOORBELL Interrupt Clear Register 0	Section 5.17
0x0210	DOORBELL1_ICSR	DOORBELL Interrupt Status Register 1	Section 5.16
0x0218	DOORBELL1_ICCR	DOORBELL Interrupt Clear Register 1	Section 5.17
0x0220	DOORBELL2_ICSR	DOORBELL Interrupt Status Register 2	Section 5.16
0x0228	DOORBELL2_ICCR	DOORBELL Interrupt Clear Register 2	Section 5.17
0x0230	DOORBELL3_ICSR	DOORBELL Interrupt Status Register 3	Section 5.16
0x0238	DOORBELL3_ICCR	DOORBELL Interrupt Clear Register 3	Section 5.17
0x0240	RX_CPPI_ICSR	RX CPPI Interrupt Status Register	Section 5.18
0x0248	RX_CPPI_ICCR	RX CPPI Interrupt Clear Register	Section 5.19
0x0250	TX_CPPI_ICSR	TX CPPI Interrupt Status Register	Section 5.20
0x0258	TX_CPPI_ICCR	TX CPPI Interrupt Clear Register	Section 5.21
0x0260	LSU_ICSR	LSU Interrupt Status Register	Section 5.22
0x0268	LSU _ICCR	LSU Interrupt Clear Register	Section 5.23
0x0270	ERR_RST_EVNT_ICSR	Error, Reset, and Special Event Interrupt Status Register	Section 5.24
0x0278	ERR_RST_EVNT_ICCR	Error, Reset, and Special Event Interrupt Clear Register	Section 5.25
0x0280	DOORBELL0_ICRR	DOORBELL0 Interrupt Condition Routing Register (bits 0 to 7)	Section 5.26
0x0284	DOORBELL0_ICRR2	DOORBELL 0 Interrupt Condition Routing Register 2 (bits 8 to 15)	Section 5.27
0x0290	DOORBELL1_ICRR	DOORBELL1 Interrupt Condition Routing Register (bits 0 to 7)	Section 5.26
0x0294	DOORBELL1_ICRR2	DOORBELL 1 Interrupt Condition Routing Register 2 (bits 8 to 15)	Section 5.27
0x02A0	DOORBELL2_ICRR	DOORBELL2 Interrupt Condition Routing Register (bits 0 to 7)	Section 5.26
0x02A4	DOORBELL2_ICRR2	DOORBELL 2 Interrupt Condition Routing Register 2 (bits 8 to 15)	Section 5.27
0x02B0	DOORBELL3_ICRR	DOORBELL3 Interrupt Condition Routing Register (bits 0 to 7)	Section 5.26
0x02B4	DOORBELL3_ICRR2	DOORBELL 3 Interrupt Condition Routing Register 2 (bits 8 to 15)	Section 5.27
0x02C0	RX_CPPI _ICRR	Receive CPPI Interrupt Condition Routing Register (0 to 7)	Section 5.28
0x02C4	RX_CPPI _ICRR2	Receive CPPI Interrupt Condition Routing Register (8 to 15)	Section 5.29
0x02D0	TX_CPPI _ICRR	Transmit CPPI Interrupt Condition Routing Register (0 to 7)	Section 5.30
0x02D4	TX_CPPI _ICRR2	Transmit CPPI Interrupt Condition Routing Register (8 to 15)	Section 5.31
0x02E0	LSU_ICRR0	LSU Interrupt Condition Routing Register 0	Section 5.32
0x02E4	LSU_ICRR1	LSU Interrupt Condition Routing Register 1	Section 5.33
0x02E8	LSU_ICRR2	LSU Interrupt Condition Routing Register 2	Section 5.34

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x02EC	LSU_ICRR3	LSU Interrupt Condition Routing Register 3	Section 5.35
0x02F0	ERR_RST_EVNT_ICRR	Error, Reset, and Special Event Interrupt Condition Routing Register	Section 5.36
0x02F4	ERR_RST_EVNT_ICRR2	Error, Reset, and Special Event Interrupt Condition Routing Register 2	Section 5.37
0x02F8	ERR_RST_EVNT_ICRR3	Error, Reset, and Special Event Interrupt Condition Routing Register 3	Section 5.38
0x0300	INTDST0_DECODE	INTDST Interrupt Status Decode Register 0	Section 5.39
0x0304	INTDST1_DECODE	INTDST Interrupt Status Decode Register 1	Section 5.39
0x0308	INTDST2_DECODE	INTDST Interrupt Status Decode Register 2	Section 5.39
0x030C	INTDST3_DECODE	INTDST Interrupt Status Decode Register 3	Section 5.39
0x0310	INTDST4_DECODE	INTDST Interrupt Status Decode Register 4	Section 5.39
0x0314	INTDST5_DECODE	INTDST Interrupt Status Decode Register 5	Section 5.39
0x0318	INTDST6_DECODE	INTDST Interrupt Status Decode Register 6	Section 5.39
0x031C	INTDST7_DECODE	INTDST Interrupt Status Decode Register 7	Section 5.39
0x0320	INTDST0_RATE_CNTL	INTDST Interrupt Rate Control Register 0	Section 5.40
0x0324	INTDST1_RATE_CNTL	INTDST Interrupt Rate Control Register 1	Section 5.40
0x0328	INTDST2_RATE_CNTL	INTDST Interrupt Rate Control Register 2	Section 5.40
0x032C	INTDST3_RATE_CNTL	INTDST Interrupt Rate Control Register 3	Section 5.40
0x0330	INTDST4_RATE_CNTL	INTDST Interrupt Rate Control Register 4	Section 5.40
0x0334	INTDST5_RATE_CNTL	INTDST Interrupt Rate Control Register 5	Section 5.40
0x0338	INTDST6_RATE_CNTL	INTDST Interrupt Rate Control Register 6	Section 5.40
0x033C	INTDST7_RATE_CNTL	INTDST Interrupt Rate Control Register 7	Section 5.40
0x040	LSU1_REG0	LSU1 Control Register 0	Section 5.41
0x0404	LSU1_REG1	LSU1 Control Register 1	Section 5.42
0x0408	LSU1_REG2	LSU1 Control Register 2	Section 5.43
0x040C	LSU1_REG3	LSU1 Control Register 3	Section 5.44
0x0410	LSU1_REG4	LSU1 Control Register 4	Section 5.45
0x0414	LSU1_REG5	LSU1 Control Register 5	Section 5.46
0x0418	LSU1_REG6	LSU1 Control Register 6	Section 5.47
0x041C	LSU1_FLOW_MASKS	Core 0 LSU Congestion Control Flow Mask Register	Section 5.48
0x0420	LSU2_REG0	LSU2 Control Register 0	Section 5.41
0x0424	LSU2_REG1	LSU2 Control Register 1	Section 5.42
0x0428	LSU2_REG2	LSU2 Control Register 2	Section 5.43
0x042C	LSU2_REG3	LSU2 Control Register 3	Section 5.44
0x0430	LSU2_REG4	LSU2 Control Register 4	Section 5.45
0x0434	LSU2_REG5	LSU2 Control Register 5	Section 5.46
0x0438	LSU2_REG6	LSU2 Control Register 6	Section 5.47
0x043C	LSU2_FLOW_MASKS1	Core 1 LSU Congestion Control Flow Mask Register	Section 5.48
0x0440	LSU3_REG0	LSU3 Control Register 0	Section 5.41

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x0444	LSU3_REG1	LSU3 Control Register 1	Section 5.42
0x0448	LSU3_REG2	LSU3 Control Register 2	Section 5.43
0x044C	LSU3_REG3	LSU3 Control Register 3	Section 5.44
0x0450	LSU3_REG4	LSU3 Control Register 4	Section 5.45
0x0454	LSU3_REG5	LSU3 Control Register 5	Section 5.46
0x0458	LSU3_REG6	LSU3 Control Register 6	Section 5.47
0x045C	LSU3_FLOW_MASKS2	Core 2 LSU Congestion Control Flow Mask Register	Section 5.48
0x0460	LSU4_REG0	LSU4 Control Register 0	Section 5.41
0x0464	LSU4_REG1	LSU4 Control Register 1	Section 5.42
0x0468	LSU4_REG2	LSU4 Control Register 2	Section 5.43
0x046C	LSU4_REG3	LSU4 Control Register 3	Section 5.44
0x0470	LSU4_REG4	LSU4 Control Register 4	Section 5.45
0x0474	LSU4_REG5	LSU4 Control Register 5	Section 5.46
0x0478	LSU4_REG6	LSU4 Control Register 6	Section 5.47
0x047C	LSU4_FLOW_MASKS3	Core 3 LSU Congestion Control Flow Mask Register	Section 5.48
0x0500	QUEUE0_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 0	Section 5.49
0x0504	QUEUE1_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 1	Section 5.49
0x0508	QUEUE2_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 2	Section 5.49
0x050C	QUEUE3_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 3	Section 5.49
0x0510	QUEUE4_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 4	Section 5.49
0x0514	QUEUE5_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 5	Section 5.49
0x0518	QUEUE6_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 6	Section 5.49
0x051C	QUEUE7_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 7	Section 5.49
0x0520	QUEUE8_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 8	Section 5.49
0x0524	QUEUE9_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 9	Section 5.49
0x0528	QUEUE10_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 10	Section 5.49
0x052C	QUEUE11_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 11	Section 5.49
0x0530	QUEUE12_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 12	Section 5.49
0x0534	QUEUE13_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 13	Section 5.49
0x0538	QUEUE14_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 14	Section 5.49
0x053C	QUEUE15_TXDMA_HDP	Queue Transmit DMA Head Descriptor Pointer Register 15	Section 5.49
0x0580	QUEUE0_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 0	Section 5.50

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x0584	QUEUE1_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 1	Section 5.50
0x0588	QUEUE2_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 2	Section 5.50
0x058C	QUEUE3_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 3	Section 5.50
0x0590	QUEUE4_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 4	Section 5.50
0x0594	QUEUE5_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 5	Section 5.50
0x0598	QUEUE6_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 6	Section 5.50
0x059C	QUEUE7_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 7	Section 5.50
0x05A0	QUEUE8_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 8	Section 5.50
0x05A4	QUEUE9_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 9	Section 5.50
0x05A8	QUEUE10_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 10	Section 5.50
0x05AC	QUEUE11_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 11	Section 5.50
0x05B0	QUEUE12_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 12	Section 5.50
0x05B4	QUEUE13_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 13	Section 5.50
0x05B8	QUEUE14_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 14	Section 5.50
0x05BC	QUEUE15_TXDMA_CP	Queue Transmit DMA Completion Pointer Register 15	Section 5.50
0x0600	QUEUE0_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 0	Section 5.51
0x0604	QUEUE1_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 1	Section 5.51
0x0608	QUEUE2_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 2	Section 5.51
0x060C	QUEUE3_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 3	Section 5.51
0x0610	QUEUE4_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 4	Section 5.51
0x0614	QUEUE5_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 5	Section 5.51
0x0618	QUEUE6_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 6	Section 5.51
0x061C	QUEUE7_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 7	Section 5.51
0x0620	QUEUE8_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 8	Section 5.51
0x0624	QUEUE9_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 9	Section 5.51
0x0628	QUEUE10_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 10	Section 5.51
0x062C	QUEUE11_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 11	Section 5.51

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x0630	QUEUE12_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 12	Section 5.51
0x0634	QUEUE13_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 13	Section 5.51
0x0638	QUEUE14_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 14	Section 5.51
0x063C	QUEUE15_RXDMA_HDP	Queue Receive DMA Head Descriptor Pointer Register 15	Section 5.51
0x0680	QUEUE0_RXDMA_CP	Queue Receive DMA Completion Pointer Register 0	Section 5.52
0x0684	QUEUE1_RXDMA_CP	Queue Receive DMA Completion Pointer Register 1	Section 5.52
0x0688	QUEUE2_RXDMA_CP	Queue Receive DMA Completion Pointer Register 2	Section 5.52
0x068C	QUEUE3_RXDMA_CP	Queue Receive DMA Completion Pointer Register 3	Section 5.52
0x0690	QUEUE4_RXDMA_CP	Queue Receive DMA Completion Pointer Register 4	Section 5.52
0x0694	QUEUE5_RXDMA_CP	Queue Receive DMA Completion Pointer Register 5	Section 5.52
0x0698	QUEUE6_RXDMA_CP	Queue Receive DMA Completion Pointer Register 6	Section 5.52
0x069C	QUEUE7_RXDMA_CP	Queue Receive DMA Completion Pointer Register 7	Section 5.52
0x06A0	QUEUE8_RXDMA_CP	Queue Receive DMA Completion Pointer Register 8	Section 5.52
0x06A4	QUEUE9_RXDMA_CP	Queue Receive DMA Completion Pointer Register 9	Section 5.52
0x06A8	QUEUE10_RXDMA_CP	Queue Receive DMA Completion Pointer Register 10	Section 5.52
0x06AC	QUEUE11_RXDMA_CP	Queue Receive DMA Completion Pointer Register 11	Section 5.52
0x06B0	QUEUE12_RXDMA_CP	Queue Receive DMA Completion Pointer Register 12	Section 5.52
0x06B4	QUEUE13_RXDMA_CP	Queue Receive DMA Completion Pointer Register 13	Section 5.52
0x06B8	QUEUE14_RXDMA_CP	Queue Receive DMA Completion Pointer Register 14	Section 5.52
0x06BC	QUEUE15_RXDMA_CP	Queue Receive DMA Completion Pointer Register 15	Section 5.52
0x0700	TX_QUEUE_TEAR_DOWN	Transmit Queue Teardown Register	Section 5.53
0x0704	TX_CPPI_FLOW_MASKS0	Transmit CPPI Supported Flow Mask Register 0	Section 5.54
0x0708	TX_CPPI_FLOW_MASKS1	Transmit CPPI Supported Flow Mask Register 1	Section 5.54
0x070C	TX_CPPI_FLOW_MASKS2	Transmit CPPI Supported Flow Mask Register 2	Section 5.54
0x0710	TX_CPPI_FLOW_MASKS3	Transmit CPPI Supported Flow Mask Register 3	Section 5.54
0x0714	TX_CPPI_FLOW_MASKS4	Transmit CPPI Supported Flow Mask Register 4	Section 5.54
0x0718	TX_CPPI_FLOW_MASKS5	Transmit CPPI Supported Flow Mask Register 5	Section 5.54

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x071C	TX_CPPI_FLOW_MASKS6	Transmit CPPI Supported Flow Mask Register 6	Section 5.54
0x0720	TX_CPPI_FLOW_MASKS7	Transmit CPPI Supported Flow Mask Register 7	Section 5.54
0x0740	RX_QUEUE_TEAR_DOWN	Receive Queue Teardown Register	Section 5.55
0x0744	RX_CPPI_CNTL	Receive CPPI Control Register	Section 5.56
0x07E0	TX_QUEUE_CNTL0	Transmit CPPI Weighted Round Robin Control Register 0	Section 5.57
0x07E4	TX_QUEUE_CNTL1	Transmit CPPI Weighted Round Robin Control Register 1	Section 5.58
0x07E8	TX_QUEUE_CNTL2	Transmit CPPI Weighted Round Robin Control Register 2	Section 5.59
0x07EC	TX_QUEUE_CNTL3	Transmit CPPI Weighted Round Robin Control Register 3	Section 5.60
0x0800	RXU_MAP_L0	MailBox-to-Queue Mapping Register L0	Section 5.61
0x0804	RXU_MAP_H0	MailBox-to-Queue Mapping Register H0	Section 5.62
0x0808	RXU_MAP_L1	MailBox-to-Queue Mapping Register L1	Section 5.61
0x080C	RXU_MAP_H1	MailBox-to-Queue Mapping Register H1	Section 5.62
0x0810	RXU_MAP_L2	MailBox-to-Queue Mapping Register L2	Section 5.61
0x0814	RXU_MAP_H2	MailBox-to-Queue Mapping Register H2	Section 5.62
0x0818	RXU_MAP_L3	MailBox-to-Queue Mapping Register L3	Section 5.61
0x081C	RXU_MAP_H3	MailBox-to-Queue Mapping Register H3	Section 5.62
0x0820	RXU_MAP_L4	MailBox-to-Queue Mapping Register L4	Section 5.61
0x0824	RXU_MAP_H4	MailBox-to-Queue Mapping Register H4	Section 5.62
0x0828	RXU_MAP_L5	MailBox-to-Queue Mapping Register L5	Section 5.61
0x082C	RXU_MAP_H5	MailBox-to-Queue Mapping Register H5	Section 5.62
0x0830	RXU_MAP_L6	MailBox-to-Queue Mapping Register L6	Section 5.61
0x0834	RXU_MAP_H6	MailBox-to-Queue Mapping Register H6	Section 5.62
0x0838	RXU_MAP_L7	MailBox-to-Queue Mapping Register L7	Section 5.61
0x083C	RXU_MAP_H7	MailBox-to-Queue Mapping Register H7	Section 5.62
0x0840	RXU_MAP_L8	MailBox-to-Queue Mapping Register L8	Section 5.61
0x0844	RXU_MAP_H8	MailBox-to-Queue Mapping Register H8	Section 5.62
0x0848	RXU_MAP_L9	MailBox-to-Queue Mapping Register L9	Section 5.61
0x084C	RXU_MAP_H9	MailBox-to-Queue Mapping Register H9	Section 5.62
0x0850	RXU_MAP_L10	MailBox-to-Queue Mapping Register L10	Section 5.61
0x0854	RXU_MAP_H10	MailBox-to-Queue Mapping Register H10	Section 5.62
0x0858	RXU_MAP_L11	MailBox-to-Queue Mapping Register L11	Section 5.61
0x085C	RXU_MAP_H11	MailBox-to-Queue Mapping Register H11	Section 5.62
0x0860	RXU_MAP_L12	MailBox-to-Queue Mapping Register L12	Section 5.61
0x0864	RXU_MAP_H12	MailBox-to-Queue Mapping Register H12	Section 5.62
0x0868	RXU_MAP_L13	MailBox-to-Queue Mapping Register L13	Section 5.61
0x086C	RXU_MAP_H13	MailBox-to-Queue Mapping Register H13	Section 5.62
0x0870	RXU_MAP_L14	MailBox-to-Queue Mapping Register L14	Section 5.61
0x0874	RXU_MAP_H14	MailBox-to-Queue Mapping Register H14	Section 5.62
0x0878	RXU_MAP_L15	MailBox-to-Queue Mapping Register L15	Section 5.61
0x087C	RXU_MAP_H15	MailBox-to-Queue Mapping Register H15	Section 5.62
0x0880	RXU_MAP_L16	MailBox-to-Queue Mapping Register L16	Section 5.61
0x0884	RXU_MAP_H16	MailBox-to-Queue Mapping Register H16	Section 5.62
0x0888	RXU_MAP_L17	MailBox-to-Queue Mapping Register L17	Section 5.61
0x088C	RXU_MAP_H17	MailBox-to-Queue Mapping Register H17	Section 5.62

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x0890	RXU_MAP_L18	MailBox-to-Queue Mapping Register L18	Section 5.61
0x0894	RXU_MAP_H18	MailBox-to-Queue Mapping Register H18	Section 5.62
0x0898	RXU_MAP_L19	MailBox-to-Queue Mapping Register L19	Section 5.61
0x089C	RXU_MAP_H19	MailBox-to-Queue Mapping Register H19	Section 5.62
0x08A0	RXU_MAP_L20	MailBox-to-Queue Mapping Register L20	Section 5.61
0x08A4	RXU_MAP_H20	MailBox-to-Queue Mapping Register H20	Section 5.62
0x08A8	RXU_MAP_L21	MailBox-to-Queue Mapping Register L21	Section 5.61
0x08AC	RXU_MAP_H21	MailBox-to-Queue Mapping Register H21	Section 5.62
0x08B0	RXU_MAP_L22	MailBox-to-Queue Mapping Register L22	Section 5.61
0x08B4	RXU_MAP_H22	MailBox-to-Queue Mapping Register H22	Section 5.62
0x08B8	RXU_MAP_L23	MailBox-to-Queue Mapping Register L23	Section 5.61
0x08BC	RXU_MAP_H23	MailBox-to-Queue Mapping Register H23	Section 5.62
0x08C0	RXU_MAP_L24	MailBox-to-Queue Mapping Register L24	Section 5.61
0x08C4	RXU_MAP_H24	MailBox-to-Queue Mapping Register H24	Section 5.62
0x08C8	RXU_MAP_L25	MailBox-to-Queue Mapping Register L25	Section 5.61
0x08CC	RXU_MAP_H25	MailBox-to-Queue Mapping Register H25	Section 5.62
0x08D0	RXU_MAP_L26	MailBox-to-Queue Mapping Register L26	Section 5.61
0x08D4	RXU_MAP_H26	MailBox-to-Queue Mapping Register H26	Section 5.62
0x08D8	RXU_MAP_L27	MailBox-to-Queue Mapping Register L27	Section 5.61
0x08DC	RXU_MAP_H27	MailBox-to-Queue Mapping Register H27	Section 5.62
0x08E0	RXU_MAP_L28	MailBox-to-Queue Mapping Register L28	Section 5.61
0x08E4	RXU_MAP_H28	MailBox-to-Queue Mapping Register H28	Section 5.62
0x08E8	RXU_MAP_L29	MailBox-to-Queue Mapping Register L29	Section 5.61
0x08EC	RXU_MAP_H29	MailBox-to-Queue Mapping Register H29	Section 5.62
0x08F0	RXU_MAP_L30	MailBox-to-Queue Mapping Register L30	Section 5.61
0x08F4	RXU_MAP_H30	MailBox-to-Queue Mapping Register H30	Section 5.62
0x08F8	RXU_MAP_L31	MailBox-to-Queue Mapping Register L31	Section 5.61
0x08FC	RXU_MAP_H31	MailBox-to-Queue Mapping Register H31	Section 5.62
0x0900	FLOW_CNTL0	Flow Control Table Entry Register 0	Section 5.63
0x0904	FLOW_CNTL1	Flow Control Table Entry Register 1	Section 5.63
0x0908	FLOW_CNTL2	Flow Control Table Entry Register 2	Section 5.63
0x090C	FLOW_CNTL3	Flow Control Table Entry Register 3	Section 5.63
0x0910	FLOW_CNTL4	Flow Control Table Entry Register 4	Section 5.63
0x0914	FLOW_CNTL5	Flow Control Table Entry Register 5	Section 5.63
0x0918	FLOW_CNTL6	Flow Control Table Entry Register 6	Section 5.63
0x091C	FLOW_CNTL7	Flow Control Table Entry Register 7	Section 5.63
0x0920	FLOW_CNTL8	Flow Control Table Entry Register 8	Section 5.63
0x0924	FLOW_CNTL9	Flow Control Table Entry Register 9	Section 5.63
0x0928	FLOW_CNTL10	Flow Control Table Entry Register 10	Section 5.63
0x092C	FLOW_CNTL11	Flow Control Table Entry Register 11	Section 5.63
0x0930	FLOW_CNTL12	Flow Control Table Entry Register 12	Section 5.63
0x0934	FLOW_CNTL13	Flow Control Table Entry Register 13	Section 5.63
0x0938	FLOW_CNTL14	Flow Control Table Entry Register 14	Section 5.63
0x093C	FLOW_CNTL15	Flow Control Table Entry Register 15	Section 5.63
0x1000	DEV_ID	Device Identity CAR	Section 5.64
0x1004	DEV_INFO	Device Information CAR	Section 5.65

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x1008	ASBLY_ID	Assembly Identity CAR	Section 5.66
0x100C	ASBLY_INFO	Assembly Information CAR	Section 5.67
0x1010	PE_FEAT	Processing Element Features CAR	Section 5.68
0x1018	SRC_OP	Source Operations CAR	Section 5.69
0x101C	DEST_OP	Destination Operations CAR	Section 5.70
0x104C	PE_LL_CTL	Processing Element Logical Layer Control CSR	Section 5.71
0x1058	LCL_CFG_HBAR	Local Configuration Space Base Address 0 CSR	Section 5.72
0x105C	LCL_CFG_BAR	Local Configuration Space Base Address 1 CSR	Section 5.73
0x1060	BASE_ID	Base Device ID CSR	Section 5.74
0x1068	HOST_BASE_ID_LOCK	Host Base Device ID Lock CSR	Section 5.75
0x106C	COMP_TAG	Component Tag CSR	Section 5.76
0x1100	SP_MB_HEAD	1x/4x LP_Serial Port Maintenance Block Header	Section 5.77
0x1120	SP_LT_CTL	Port Link Time-Out Control CSR	Section 5.78
0x1124	SP_RT_CTL	Port Response Time-Out Control CSR	Section 5.79
0x113C	SP_GEN_CTL	Port General Control CSR	Section 5.80
0x1140	SP0_LM_REQ	Port 0 Link Maintenance Request CSR	Section 5.81
0x1144	SP0_LM_RESP	Port 0 Link Maintenance Response CSR	Section 5.82
0x1148	SP0_ACKID_STAT	Port 0 Local AckID Status CSR	Section 5.83
0x1158	SP0_ERR_STAT	Port 0 Error and Status CSR	Section 5.84
0x115C	SP0_CTL	Port 0 Control CSR	Section 5.85
0x1160	SP1_LM_REQ	Port 1 Link Maintenance Request CSR	Section 5.81
0x1164	SP1_LM_RESP	Port 1 Link Maintenance Response CSR	Section 5.82
0x1168	SP1_ACKID_STAT	Port 1 Local AckID Status CSR	Section 5.83
0x1178	SP1_ERR_STAT	Port 1 Error and Status CSR	Section 5.84
0x117C	SP1_CTL	Port 1 Control CSR	Section 5.85
0x1180	SP2_LM_REQ	Port 2 Link Maintenance Request CSR	Section 5.81
0x1184	SP2_LM_RESP	Port 2 Link Maintenance Response CSR	Section 5.82
0x1188	SP2_ACKID_STAT	Port 2 Local AckID Status CSR	Section 5.83
0x1198	SP2_ERR_STAT	Port 2 Error and Status CSR	Section 5.84
0x119C	SP2_CTL	Port 2 Control CSR	Section 5.85
0x11A0	SP3_LM_REQ	Port 3 Link Maintenance Request CSR	Section 5.81
0x11A4	SP3_LM_RESP	Port 3 Link Maintenance Response CSR	Section 5.82
0x11A8	SP3_ACKID_STAT	Port 3 Local AckID Status CSR	Section 5.83
0x11B8	SP3_ERR_STAT	Port 3 Error and Status CSR	Section 5.84
0x11BC	SP3_CTL	Port 3 Control CSR	Section 5.85
0x2000	ERR_RPT_BH	Error Reporting Block Header	Section 5.86
0x2008	ERR_DET	Logical/Transport Layer Error Detect CSR	Section 5.87
0x200C	ERR_EN	Logical/Transport Layer Error Enable CSR	Section 5.88
0x2010	H_ADDR_CAPT	Logical/Transport Layer High Address Capture CSR	Section 5.89
0x2014	ADDR_CAPT	Logical/Transport Layer Address Capture CSR	Section 5.90
0x2018	ID_CAPT	Logical/Transport Layer Device ID Capture CSR	Section 5.91
0x201C	CTRL_CAPT	Logical/Transport Layer Control Capture CSR	Section 5.92
0x2028	PW_TGT_ID	Port-Write Target Device ID CSR	Section 5.93
0x2040	SP0_ERR_DET	Port 0 Error Detect CSR	Section 5.94
0x2044	SP0_RATE_EN	Port 0 Error Enable CSR	Section 5.95

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x2048	SP0_ERR_ATTR_CAPT_DBG0	Port 0 Attributes Error Capture CSR 0	Section 5.96
0x204C	SP0_ERR_CAPT_DBG1	Port 0 Packet/Control Symbol Error Capture CSR 1	Section 5.97
0x2050	SP0_ERR_CAPT_DBG2	Port 0 Packet/Control Symbol Error Capture CSR 2	Section 5.98
0x2054	SP0_ERR_CAPT_DBG3	Port 0 Packet/Control Symbol Error Capture CSR 3	Section 5.99
0x2058	SP0_ERR_CAPT_DBG4	Port 0 Packet/Control Symbol Error Capture CSR 4	Section 5.100
0x2068	SP0_ERR_RATE	Port 0 Error Rate CSR 0	Section 5.101
0x206C	SP0_ERR_THRESH	Port 0 Error Rate Threshold CSR	Section 5.102
0x2080	SP1_ERR_DET	Port 1 Error Detect CSR	Section 5.94
0x2084	SP1_RATE_EN	Port 1 Error Enable CSR	Section 5.95
0x2088	SP1_ERR_ATTR_CAPT_DBG0	Port 1 Attributes Error Capture CSR 0	Section 5.96
0x208C	SP1_ERR_CAPT_DBG1	Port 1 Packet/Control Symbol Error Capture CSR 1	Section 5.97
0x2090	SP1_ERR_CAPT_DBG2	Port 1 Packet/Control Symbol Error Capture CSR 2	Section 5.98
0x2094	SP1_ERR_CAPT_DBG3	Port 1 Packet/Control Symbol Error Capture CSR 3	Section 5.99
0x2098	SP1_ERR_CAPT_DBG4	Port 1 Packet/Control Symbol Error Capture CSR 4	Section 5.100
0x20A8	SP1_ERR_RATE	Port 1 Error Rate CSR	Section 5.101
0x20AC	SP1_ERR_THRESH	Port 1 Error Rate Threshold CSR	Section 5.102
0x20C0	SP2_ERR_DET	Port 2 Error Detect CSR	Section 5.94
0x20C4	SP2_RATE_EN	Port 2 Error Enable CSR	Section 5.95
0x20C8	SP2_ERR_ATTR_CAPT_DBG0	Port 2 Attributes Error Capture CSR 0	Section 5.96
0x20CC	SP2_ERR_CAPT_DBG1	Port 2 Packet/Control Symbol Error Capture CSR 1	Section 5.97
0x20D0	SP2_ERR_CAPT_DBG2	Port 2 Packet/Control Symbol Error Capture CSR 2	Section 5.98
0x20D4	SP2_ERR_CAPT_DBG3	Port 2 Packet/Control Symbol Error Capture CSR 3	Section 5.99
0x20D8	SP2_ERR_CAPT_DBG4	Port 2 Packet/Control Symbol Error Capture CSR 4	Section 5.100
0x20E8	SP2_ERR_RATE	Port 2 Error Rate CSR	Section 5.101
0x20EC	SP2_ERR_THRESH	Port 2 Error Rate Threshold CSR	Section 5.102
0x2100	SP3_ERR_DET	Port 3 Error Detect CSR	Section 5.94
0x2104	SP3_RATE_EN	Port 3 Error Enable CSR	Section 5.95
0x2108	SP3_ERR_ATTR_CAPT_DBG0	Port 3 Attributes Error Capture CSR 0	Section 5.96
0x210C	SP3_ERR_CAPT_DBG1	Port 3 Packet/Control Symbol Error Capture CSR 1	Section 5.97
0x2110	SP3_ERR_CAPT_DBG2	Port 3 Packet/Control Symbol Error Capture CSR 2	Section 5.98
0x2114	SP3_ERR_CAPT_DBG3	Port 3 Packet/Control Symbol Error Capture CSR 3	Section 5.99
0x2118	SP3_ERR_CAPT_DBG4	Port 3 Packet/Control Symbol Error Capture CSR 4	Section 5.100

Table 28. Serial Rapid IO (SRIO) Registers (continued)

Offset	Acronym	Register Description	Section
0x2128	SP3_ERR_RATE	Port 3 Error Rate CSR	Section 5.101
0x212C	SP3_ERR_THRESH	Port 3 Error Rate Threshold CSR	Section 5.102
0x12000	SP_IP_DISCOVERY_TIMER	Port IP Discovery Timer in 4x mode	Section 5.103
0x12004	SP_IP_MODE	Port IP Mode CSR	Section 5.104
0x12008	IP_PRESCAL	Serial Port IP Prescalar	Section 5.105
0x12010	SP_IP_PW_IN_CAPT0	Port-Write-In Capture CSR Register 0	Section 5.106
0x12014	SP_IP_PW_IN_CAPT1	Port-Write-In Capture CSR Register 1	Section 5.106
0x12018	SP_IP_PW_IN_CAPT2	Port-Write-In Capture CSR Register 2	Section 5.106
0x1201C	SP_IP_PW_IN_CAPT3	Port-Write-In Capture CSR Register 3	Section 5.106
0x14000	SP0_RST_OPT	Port 0 Reset Option CSR	Section 5.107
0x14004	SP0_CTL_INDEP	Port 0 Control Independent Register	Section 5.108
0x14008	SP0_SILENCE_TIMER	Port 0 Silence Timer Register	Section 5.109
0x1400C	SP0_MULT_EVNT_CS	Port 0 Multicast-Event Control Symbol Request Register	Section 5.110
0x14014	SP0_CS_TX	Port 0 Control Symbol Transmit Register	Section 5.111
0x14100	SP1_RST_OPT	Port 1 Reset Option CSR	Section 5.107
0x14104	SP1_CTL_INDEP	Port 1 Control Independent Register	Section 5.108
0x14108	SP1_SILENCE_TIMER	Port 1 Silence Timer Register	Section 5.109
0x1410C	SP1_MULT_EVNT_CS	Port 1 Multicast-Event Control Symbol Request Register	Section 5.110
0x14114	SP1_CS_TX	Port 1 Control Symbol Transmit Register	Section 5.111
0x14200	SP2_RST_OPT	Port 2 Reset Option CSR	Section 5.107
0x14204	SP2_CTL_INDEP	Port 2 Control Independent Register	Section 5.108
0x14208	SP2_SILENCE_TIMER	Port 2 Silence Timer Register	Section 5.109
0x1420C	SP2_MULT_EVNT_CS	Port 2 Multicast-Event Control Symbol Request Register	Section 5.110
0x14214	SP2_CS_TX	Port 2 Control Symbol Transmit Register	Section 5.111
0x14300	SP3_RST_OPT	Port 3 Reset Option CSR	Section 5.107
0x14304	SP3_CTL_INDEP	Port 3 Control Independent Register	Section 5.108
0x14308	SP3_SILENCE_TIMER	Port 3 Silence Timer Register	Section 5.109
0x1430C	SP3_MULT_EVNT_CS	Port 3 Multicast-Event Control Symbol Request Register	Section 5.110
0x14314	SP3_CS_TX	Port 3 Control Symbol Transmit Register	Section 5.111

5.2 Peripheral Identification Register (PID)

The peripheral identification register (PID) is a constant register that contains the ID and ID revision number for that peripheral. The PID stores version information used to identify the peripheral. All bits within this register are read-only (writes have no effect) meaning that the values within this register should be hard-coded with the appropriate values and must not change from their reset state.

Figure 58. Peripheral ID Register (PID)

31-24	23-16
Reserved	TYPE
R-0x00	R-0x01

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
CLASS	REV
R-0x0A	R-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 29. Peripheral ID Register (PID) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved		Reserved
23-16	TYPE		Peripheral type: Identifies the type of the peripheral RIO
15-8	CLASS		Peripheral class: Identifies the class Switch Fabric
7-0	REV		Peripheral revision: Identifies the revision of the peripheral. This value should begin at 0x01 and be incremented each time the design is revised .

5.3 Peripheral Control Register (PCR)

The peripheral control register (PCR) contains a bit that enables or disables the entire peripheral and one bit for every module within the peripheral where this level of control is desired. The module control bits can only be written when the peripheral itself is enabled. In addition, the PCR has emulation control bits free and soft, which control the peripheral behavior during emulation halts.

Figure 59. Peripheral Control Register (PCR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-3	2	1	0
Reserved	PEREN	SOFT	FREE
R-0x00	RW-0x00	RW-0x00	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 30. Peripheral Control Register (PCR) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved		Reserved
2	PEREN		Peripheral Enable. Controls the flow of data in the logical layer of the peripheral. As an initiator, it will prevent TX transaction generation and as a target, it will disable incoming requests. This should be the last enable bit to toggle when bringing the device out of reset to begin normal operation.
		0b	Disables data flow control
		1b	Enables data flow control
1	SOFT		Emulation Control - SOFT bit
0	FREE		Emulation Control - FREE bit

5.4 Peripheral Settings Control Register (PER_SET_CNTL)

Figure 60. Peripheral Settings Control Register (PER_SET_CNTL)

31-27	26	25	24	23-21	20-18	17-16
Reserved	SW_MEM_SLEEP_OVERRIDE	LOOPBACK	BOOT_COMPLETE	Reserved	TX_PRI2_WM	TX_PRI1_WM
R-0x00	RW-0x01	RW-0x00	RW-0x00	R-0x00	RW-0x01	RW-0x02

LEGEND: R = Read only; -*n* = value after reset

15	14-12	11-9	8	7-4	3	2	1	0
TX_PRI1_WM	TX_PRI0_WM	CBA_TRANS_PRI	1X_MODE	PRESCALER_SELECT	ENPLL4	ENPLL3	ENPLL2	ENPLL1
RW-0x02	RW-0x03	RW-0x00	RW-0x00	RW-0x0000	RW-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 31. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions

Bit	Field	Value	Description
31-27	Reserved		Reserved
26	SW_MEM_SLEEP_OVERRIDE		Software Memory Sleep Override
		0b	Memories are put in sleep mode while in shutdown
		1b	Memories are not put in sleep mode while in shutdown
25	LOOPBACK		Loopback mode.
		0b	Normal operation
		1b	Loop back mode. Transmit data to receive on the same port. Packet data is looped back in the digital domain before the SERDES macros.
24	BOOT_COMPLETE		Controls ability to write any register during initialization. It also includes read only registers during normal mode of operation that have application defined reset value.
		0b	Write to read only registers enabled
		1b	Write to read only registers disabled. Usually the boot_complete is asserted once after reset to define power on configuration.
23-21	Reserved		Reserved
20-18	TX_PRI2_WM		Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 2 packets across the UDI interface. This is valid for all ports in 1X mode only. Required buffer count for transmit credit threshold 2 value (TX_PRI2_WM): • 000→8, 7, 6, 5, 4, 3, 2, 1 • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8

Table 31. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
17-15	TX_PRI1_WM		Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 1 packets across the UDI interface. This is valid for all ports in 1X mode only. Required buffer count for transmit credit threshold 1 value TX_PRI1_WM: • 000→8, 7, 6, 5, 4, 3, 2, 1 • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8
14-12	TX_PRI0_WM		Transmit credit threshold. Sets the required number of logical layer TX buffers needed to send priority 0 packets across the UDI interface. This is valid for all ports in 1X mode only. Required buffer count for transmit credit threshold 0 value TX_PRI0_WM: • 000→8, 7, 6, 5, 4, 3, 2, 1 • 001→8, 7, 6, 5, 4, 3, 2 • 010→8, 7, 6, 5, 4, 3 • 011→8, 7, 6, 5, 4 • 100→8, 7, 6, 5 • 101→8, 7, 6 • 110→8, 7 • 111→8
11-9	CBA_TRANS_PRI		VBUS transaction priority. 000b – Highest Priority … 111b – Lowest Priority.
8	1X_MODE		This register bit determines the UDI buffering setup (priority versus port).
		0b	UDI buffers are priority based
		1b	UDI buffers are port based. This mode must be selected when using more than one 1X port
7-4	PRESCALER_SELECT		Internal frequency prescaler, used to drive the request to response timers. These 4 bits are the prescaler reload value allowing division of the DMA clock by a range from 1 up to 16. Setting should reflect the device DMA frequency.
		0000b	Sets the internal clock frequency Min 44.7 and Max 89.5
		0001b	Sets the internal clock frequency Min 89.5 and Max 179.0
		0010b	Sets the internal clock frequency Min 134.2 and Max 268.4
		0011b	Sets the internal clock frequency Min 180.0 and Max 360.0
		0100b	Sets the internal clock frequency Min 223.7 and Max 447.4
		0101b	Sets the internal clock frequency Min 268.4 and Max 536.8
		0110b	Sets the internal clock frequency Min 313.2 and Max 626.4
		0111b	Sets the internal clock frequency Min 357.9 and Max 715.8
		1000b	sets the internal clock frequency Min 402.7 and Max 805.4
		1001b	Sets the internal clock frequency Min 447.4 and Max 894.8
		1010b	Sets the internal clock frequency Min 492.1 and Max 984.2
		1011b	Sets the internal clock frequency Min 536.9 and Max 1073.8
		1100b	Sets the internal clock frequency Min 581.6 and Max 1163.2
		1101b	Sets the internal clock frequency Min 626.3 and Max 1252.6
		1110b	Sets the internal clock frequency Min 671.1 and Max 1342.2
		1111b	Sets the internal clock frequency Min 715.8 and Max 1431.6
3	ENPLL4		Drives SERDES Macro 4 PLL Enable signal
		0b	Disables macro 4 PLL
		1b	Enables macro 4 PLL

Table 31. Peripheral Settings Control Register (PER_SET_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
2	ENPLL3		Drives SERDES Macro 3 PLL Enable signal
		0b	Disables macro 3 PLL
		1b	Enables macro 3 PLL
1	ENPLL2		Drives SERDES Macro 2 PLL Enable signal
		0b	Disables macro 2 PLL
		1b	Enables macro 2 PLL
0	ENPLL1		Drives SERDES Macro 1 PLL Enable signal
		0b	Disables macro 1 PLL
		1b	Enables macro 1 PLL

5.5 Peripheral Global Enable Register (GBL_EN)

Figure 61. Peripheral Global Enable Register (GBL_EN)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-1	0
Reserved	EN
R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 32. Peripheral Global Enable Register (GBL_EN) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved		Reserved
0	EN		Controls reset to all clock domains within the peripheral
		0b	Peripheral to be disabled (held in reset, clocks disabled)
		1b	Peripheral to be enabled

5.6 Peripheral Global Enable Status Register (GBL_EN_STAT)

Figure 62. Peripheral Global Enable Status Register (GBL_EN_STAT)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	1	0
Reserved		GBL_EN_STAT
R-0x00		R-Undefined

LEGEND: R = Read only; -*n* = value after reset

Table 33. Peripheral Global Enable Status Register (GBL_EN_STAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved		Reserved
0	GBL_EN_STAT		Indicates state of GBL_EN reset signal
		0	Peripheral in reset and all clocks are off
		1	Peripheral enabled and clocking

5.7 Block n Enable Register (BLKn_EN)

There are nine of these registers, one for each of nine logical blocks in the peripheral.

Figure 63. Block *n* Enable Register (BLK*n*_EN)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-1	0
Reserved	EN
R-0x00	RW-Undefined

LEGEND: R = Read only; -*n* = value after reset

Table 34. Block *n* Enable Register (BLK*n*_EN) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved		Reserved
0	EN		Controls reset to *n*th (0 to 8) clock/logical domain.
		0	Logical block *n* disabled (held in reset, clocks disabled)
		1	Logical block *n* enabled

5.8 Block n Enable Status Register (BLKn_EN_STAT)

There are nine of these registers, one for each of nine logical blocks in the peripheral.

Figure 64. Block *n* Enable Status Register (BLK*n*_EN_STAT)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-1	0
Reserved	EN_STAT
R-0x00	R-Undefined

LEGEND: R = Read only; -*n* = value after reset

Table 35. Block *n* Enable Status Register (BLK*n*_EN_STAT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved		Reserved
0	EN_STAT		Indicates state of BLK*n*_EN reset signal
		0	Logical block *n* disabled, in reset and clock is off
		1	Logical block *n* enabled and clocking

5.9 RapidIO DEVICEID1 Register (DEVICEID_REG1)

Figure 65. RapidIO DEVICEID1 Register (DEVICEID_REG1)

31-24	23-16
Reserved	8BNODEID
R-0x0000	RW-0x00FF

LEGEND: R = Read only; -*n* = value after reset

15-0
16BNODEID
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

Table 36. RapidIO DEVICEID1 Register (DEVICEID_REG1) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved		Reserved
23-16	8BNODEID		This value is equal to the value of the RapidIO Base Device ID CSR. The CPU must read the CSR value and set this register, so that outgoing packets contain the correct SOURCEID value
15-0	16BNODEID		This value is equal to the value of the RapidIO Base Device ID CSR. The CPU must read the CSR value and set this register, so that outgoing packets contain the correct SOURCEID value

5.10 RapidIO DEVICEID2 Register (DEVICEID_REG2)

Figure 66. RapidIO DEVICEID2 Register (DEVICEID_REG2)

31-24	23-16
Reserved	8BNODEID
R-0x0000	RW-0x00FF

LEGEND: R = Read only; -*n* = value after reset

15-0
16BNODEID
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

Table 37. RapidIO DEVICEID2 Register (DEVICEID_REG2) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved		Reserved
23-16	8BNODEID		This is a secondary supported DeviceID checked against an incoming packet's DestID field. Typically used for Multi-cast support.
15-0	16BNODEID		This is a secondary supported DeviceID checked against an incoming packet's DestID field. Typically used for Multi-cast support

5.11 Packet Forwarding Register n for 16b DeviceIDs (PF_16B_CNTLn)

There are four of these registers, to support four ports.

Figure 67. Packet Forwarding Register *n* for 16b DeviceIDs (PF_16B_CNTL*n*)

31-16
16BIT_DEVID_UP_BOUND
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

15-0
16BIT_DEVID_LOW_BOUND
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

Table 38. Packet Forwarding Register *n* for 16b DeviceIDs (PF_16B_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-16	16BIT_DEVID_UP_BOUND		Upper 16b DeviceID boundary. DestID above this range cannot use the table entry.
15-0	16BIT_DEVID_LOW_BOUND		Lower 16b DeviceID boundary. DestID lower than this number cannot use the table entry.

5.12 Packet Forwarding Register n for 8b DeviceIDs (PF_8B_CNTLn)

There are four of these registers, to support four ports.

Figure 68. Packet Forwarding Register *n* for 8b DeviceIDs (PF_8B_CNTL*n*)

31-18	17-16
Reserved	OUT_BOUND_PORT
R-0x00	RW-0x03

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
8BIT_DEVID_UP_BOUND	8BIT_DEVID_LOW_BOUND
RW-0xFF	RW-0xFF

LEGEND: R = Read only; -*n* = value after reset

Table 39. Packet Forwarding Register *n* for 8b DeviceIDs (PF_8B_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved		Reserved
17-16	OUT_BOUND_PORT		Output port number for packets whose DestID falls within the 8b or 16b range for this table entry.
15-8	8BIT_DEVID_UP_BOUND		Upper 8b DeviceID boundary. DestID above this range cannot use the table entry.
7-0	8BIT_DEVID_LOW_BOUND		Lower 8b DeviceID boundary. DestID lower than this number cannot use the table entry.

5.13 SERDES Receive Channel Configuration Registers n (SERDES_CFGRXn_CNTL)

There are four of these registers, to support four ports.

Figure 69. SERDES Receive Channel Configuration Registers *n* (SERDES_CFGRX*n*_CNTL)

31	26	25	24	23	22	19	18	16
Reserved		Reserved	Reserved	Reserved	EQ		CDR	
R-0		R/W-0	R/W-0	R-0	R/W-0		R/W-0	

15	14	13	12	11	10	8	7	6	5	4	2	1	0
LOS		ALIGN		Reserved	TERM		INVPAIR	RATE		BUSWIDTH		Reserved	ENRX
R/W-0		R/W-0		R-0	R/W-0		R/W-0	R/W-0		R/W-0		R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

Table 40. SERDES Receive Channel Configuration Registers *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31:26	Reserved		Reserved.
25	Reserved		Reserved, keep as zero during writes to this register.
24	Reserved		Reserved, keep as zero during writes to this register.
23	Reserved		Reserved.
19:22	EQ		Equalizer. Enables and configures the adaptive equalizer to compensate for loss in the transmission media. For values, see Table 41.
18:16	CDR		Clock/data recovery. Configures the clock/data recovery algorithm.
		000	First order. Phase offset tracking up to ±488 ppm.
		001	Second order. Highest precision frequency offset matching but poorest response to changes in frequency offset, and longest lock time. Suitable for use in systems with fixed frequency offset.
		010	Second order. Medium precision frequency offset matching, frequency offset change response and lock time.
		011	Second order. Best response to changes in frequency offset and fastest lock time, but lowest precision frequency offset matching. Suitable for use in systems with spread spectrum clocking.
		100	First order with fast lock. Phase offset tracking up to ±1953 ppm in the presence of ..10101010.. training pattern, and ±448 ppm otherwise.
		101	Second order with fast lock. As per setting 001, but with improved response to changes in frequency offset when not close to lock.
		110	Second order with fast lock. As per setting 010, but with improved response to changes in frequency offset when not close to lock.
		111	Second order with fast lock. As per setting 011, but with improved response to changes in frequency offset when not close to lock.
15:14	LOS		Loss of signal. Enables loss of signal detection with 2 selectable thresholds.
		00	Disabled. Loss of signal detection disabled.
		01	High threshold. Loss of signal detection threshold in the range 85 to 195mV$_{dfpp}$. This setting is suitable for Infiniband.
		10	Low threshold. Loss of signal detection threshold in the range 65-175mV$_{dfpp}$. This setting is suitable for PCI-E and S-ATA.
		11	Reserved
13:12	ALIGN		Symbol alignment. Enables internal or external symbol alignment.
		00	Alignment disabled. No symbol alignment will be performed while this setting is selected, or when switching to this selection from another.
		01	Comma alignment enabled. Symbol alignment will be performed whenever a misaligned comma symbol is received.
		10	Alignment jog. The symbol alignment will be adjusted by one bit position when this mode is selected (i.e., CFGRX[13:12] changes from 0x to 1x).
		11	Reserved

Table 40. SERDES Receive Channel Configuration Registers *n* (SERDES_CFGRX*n*_CNTL) Field Descriptions (continued)

Bit	Field	Value	Description
11	Reserved		Reserved.
10:8	TERM		Termination. Selects input termination options suitable for a variety of AC or DC coupled scenarios.
		000	Common point connected to VDDT. This configuration is for DC coupled systems using CML transmitters. The common mode voltage is determined jointly by both the receiver and the transmitter. Common mode termination is via a 50pF capacitor to VSSA.
		001	Common point set to 0.8 VDDT. This configuration is for AC coupled systems using CML transmitters. The transmitter has no effect on the receiver common mode, which is set to optimize the input sensitivity of the receiver. Common mode termination is via a 50pF capacitor to VSSA.
		010	Reserved
		011	Common point floating. This configuration is for DC coupled systems that require the common mode voltage to be determined by the transmitter only. These are typically not CML. Common mode termination is via a 50pF capacitor to VSSA.
		1xx	Reserved
7	INVPAIR		Invert polarity. Inverts polarity of RXP*I* and RXN*n*.
		0	Normal polarity. RXP*n* considered to be positive data and RXN*i* negative.
		1	Inverted polarity. RXP*n* considered to be negative data and RXN*n* positive.
6:5	RATE		Operating rate. Selects full, half or quarter rate operation.
		00	Full rate. Two data samples taken per PLL output clock cycle.
		01	Half rate. One data sample taken per PLL output clock cycle.
		10	Quarter rate. One data sample taken every two PLL output clock cycles.
		11	Reserved
4:2	BUS-WIDTH		Bus width. Selects the width of the parallel interface (10 or 8 bit).
		000	10-bit operation. Data is output on RD*n*[9:0]. RXBCLK[*n*] period is 10 bit periods (4 high, 6 low).
		001	8-bit operation. Data is output on RD*n*[7:0]. RXBCLK[*n*] period is 8 bit periods (4 high, 4 low). RD*n*[9:8] will replicate bits [1:0] from the previous byte.
		01x	Reserved
		1xx	Reserved
1	Reserved		Reserved, keep as zero during writes to this register.
0	ENRX		Enable receiver. Enables this receiver when high.
		0	Disable
		1	Enable

Table 41. EQ Bits

CFGRX[22:19]	Low Freq Gain	Zero Freq (at e_{28} (min))
0000	Maximum	-
0001	Adaptive	Adaptive
001x	Reserved	
01xx	Reserved	
1000	Adaptive	1084MHz
1001		805MHz
1010		573MHz
1011		402MHz
1100		304MHz
1101		216MHz
1110		156MHz
1111		135MHz

5.14 SERDES Transmit Channel Configuration Registers n (SERDES_CFGTXn_CNTL)

There are four of these registers, to support four ports.

Figure 70. SERDES Transmit Channel Configuration Registers *n* (SERDES_CFGTX*n*_CNTL)

31	17	16
Reserved		ENFTP
R-0		R/W-0

15 12	11 9	8	7	6 5	4 2	1	0
DE	SWING	CM	INVPAIR	RATE	BUSWIDTH	Reserved	ENTX
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R = Read, W = Write, n = value at reset

Table 42. SERDES Transmit Channel Configuration Registers *n* (SERDES_CFGTX*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31:17	Reserved		Reserved.
16	ENFTP		Enable fixed TXBCKLIN*n* phase. Enables fixed phase relationship of TXBCLKIN*n* with respect to TXBCLK*n*.
		0	Arbitrary phase. No required phase relationship between TXBCLKIN*n* and TXBCLK*n*.
		1	Fixed phase. Requires direct connection of TXBCLK*n* to TXBCLKIN*n* using a minimum length net without buffers.
15:12	DE		De-emphasis. Selects one of 15 output de-emphasis settings from 4.76 to 71.42%. See Table 44.
11:9	SWING		Output swing. Selects one of 8 outputs amplitude settings between 125 and 1250mV$_{dfpp}$. See Table 43.
8	CM		Common mode. Adjusts the common mode to suit the termination at the attached receiver.
		0	Normal common mode. Common mode not adjusted.
		1	Raised common mode. Common mode raised by 5% of e_{54}.
7	INVPAIR		Invert polarity. Inverts polarity of TXP*n* and TXN*n*.
		0	Normal polarity. TXP*n* considered to be positive data and TXN*n* negative.
		1	Inverted polarity. TXP*n* considered to be negative data and TXN*n* positive.
6:5	RATE		Operating rate. Selects full, half or quarter rate operation.
		00	Full rate. Two data samples taken per PLL output clock cycle.
		01	Half rate. One data sample taken per PLL output clock cycle.
		10	Quarter rate. One data sample taken every two PLL output clock cycles.
		11	Reserved
4:2	BUS-WIDTH		Bus width. Selects the width of the parallel interface (10 or 8 bit).
		000	10-bit operation. Data is input on TD*n*[9:0]. TXBCLK*n* period is 10 bit periods (4 high, 6 low).
		001	8-bit operation. Data is input on TD*n*[9:2]. TXBCLK*n* period is 8 bit periods (4 high, 4 low). TD*n*[1:0] are ignored.
		01x	Reserved
		1xx	Reserved
1	Reserved		Reserved
0	ENTX		Enable transmitter. Enables this transmitter when high.

Table 43. SWING Bits

CFGTX[11:9]	Amplitude (mV_{dfpp})
000	125
001	250
010	500
011	625
100	750
101	1000
110	1125
111	1250

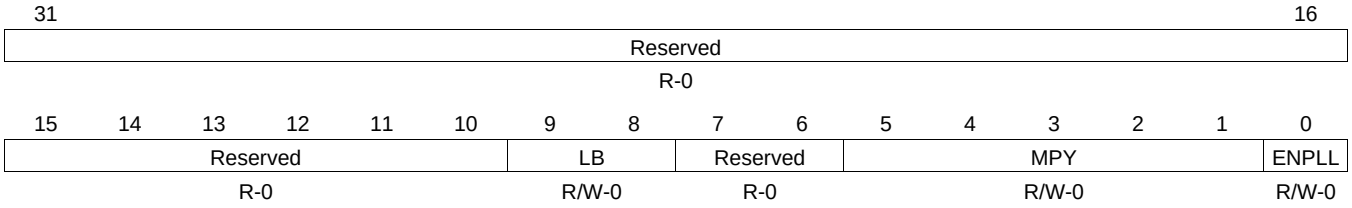
Table 44. DE Bits

CFGTX[15:12]	Amplitude Reduction	
	%	dB
0000	0	0
0001	4.76	-0.42
0010	9.52	-0.87
0011	14.28	-1.34
0100	19.04	-1.83
0101	23.8	-2.36
0110	28.56	-2.92
0111	33.32	-3.52
1000	38.08	-4.16
1001	42.85	-4.86
1010	47.61	-5.61
1011	52.38	-6.44
1100	57.14	-7.35
1101	61.9	-8.38
1110	66.66	-9.54
1111	71.42	-10.87

5.15 SERDES Macro Configuration Register n (SERDES_CFGn_CNTL)

There are four of these registers, to support four ports.

Figure 71. SERDES Macros CFG (0-3) Registers (SERDES_CFG*n*_CNTL)



LEGEND: R = Read, W = Write, n = value at reset

Table 45. SERDES Macros CFG (0-3) Registers (SERDES_CFG*n*_CNTL) Field Descriptions

Bit	Field	Value	Description
31:10	Reserved		Reserved.
9:8	LB		Loop bandwidth. Specify loop bandwidth settings.
		00	Frequency dependent bandwidth. The PLL bandwidth is set to a twelfth of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$.
		01	Reserved
		10	Low bandwidth. The PLL bandwidth is set to a twentieth of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$, or 3MHz (whichever is larger).
		11	High bandwidth. The PLL bandwidth is set to a eighth of the frequency of RIOCLK/$\overline{\text{RIOCLK}}$.
7:6	Reserved		Reserved.
5:1	MPY		PLL multiply. Select PLL multiply factors between 4 and 60.
		0000	4x
		0001	5x
		0010	6x
		0011	Reserved
		0100	8x
		0101	10x
		0110	12x
		0111	12.5x
		1000	15x
		1001	20x
		1010	25x
		1011	Reserved
		1100	Reserved
		1101	50x
		1110	60x
		1111	Reserved
0	ENPLL		Enable PLL. Enables the PLL.

5.16 DOORBELLn Interrupt Status Register (DOORBELLn_ICSR)

Each of the four doorbells is supported by a register of this type.

Figure 72. DOORBELL*n* Interrupt Status Register (DOORBELL*n*_ICSR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICS (0-15)
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 46. DOORBELL*n* Interrupt Status Register (DOORBELL*n*_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICS (0-15)		Doorbell *n* (0 to 3) interrupt condition status bits

5.17 DOORBELLn Interrupt Clear Register (DOORBELLn_ICCR)

Each of the four doorbells is supported by a register of this type.

Figure 73. DOORBELL*n* Interrupt Clear Register (DOORBELL*n*_ICCR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICC (0-15)
W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 47. DOORBELL*n* Interrupt Clear Register (DOORBELL*n*_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICC (0-15)		Doorbell *n* (0 to 3) interrupt clear bits

5.18 RX CPPI Interrupt Status Register (RX_CPPI_ICSR)

Figure 74. RX CPPI Interrupt Status Register (RX_CPPI_ICSR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICS (0-15)
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 48. RX CPPI Interrupt Status Register (RX_CPPI_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICS (0-15)		RX CPPI Interrupt, Buffer descriptor Queue 0 to 15

5.19 RX CPPI Interrupt Clear Register (RX_CPPI_ICCR)

Figure 75. RX CPPI Interrupt Clear Register (RX_CPPI_ICCR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICC (0-15)
W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 49. RX CPPI Interrupt Clear Register (RX_CPPI_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICC (0-15)		RX CPPI Interrupt clear, Buffer descriptor Queue 0 to 15

5.20 TX CPPI Interrupt Status Register (TX_CPPI_ICSR)

Figure 76. TX CPPI Interrupt Status Register (TX_CPPI_ICSR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICS (0-15)
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 50. TX CPPI Interrupt Status Register (TX_CPPI_ICSR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICS (0-15)		TX CPPI Interrupt, Buffer descriptor Queue 0 to 15

5.21 TX CPPI Interrupt Clear Register (TX_CPPI_ICCR)

Figure 77. TX CPPI Interrupt Clear Register (TX_CPPI_ICCR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICC (0-15)
W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 51. TX CPPI Interrupt Clear Register (TX_CPPI_ICCR) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	ICC (0-15)		TX CPPI Interrupt clear, Buffer descriptor Queue 0 to 15

5.22 LSU Status Interrupt Register (LSU_ICSR)

Figure 78. LSU Status Interrupt Register (LSU_ICSR)

31-16
ICS(31-16)
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICS(15-0)
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 52. LSU Status Interrupt Register (LSU_ICSR) Field Descriptions

Bit	Field	Value	Description
31-0	ICS(31-0)		Load/Store module interrupt condition status bits

5.23 LSU Clear Interrupt Register (LSU _ICCR)

Figure 79. LSU Clear Interrupt Register (LSU _ICCR)

31-16
ICC(31-16)
W-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICC(15-0)
W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 53. LSU Clear Interrupt Register (LSU _ICCR) Field Descriptions

Bit	Field	Value	Description
31-0	ICS(31-0)		Load/Store module interrupt clear bits

5.24 Error, Reset, and Special Event Status Interrupt Register (ERR_RST_EVNT_ICSR)

Figure 80. Error, Reset, and Special Event Status Interrupt Register (ERR_RST_EVNT_ICSR)

31-17	16
Reserved	ICS16
R-0x00	R/W-0x00

LEGEND: R = Read only; -*n* = value after reset

15-12	11	10	9	8	7-3	2	1	0
Reserved	ICS11	ICS10	ICS9	ICS8	Reserved	ICS2	ICS1	ICS0
R-0x00	R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00	R-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 54. Error, Reset, and Special Event Status Interrupt Register (ERR_RST_EVNT_ICSR) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved		Reserved
16	ICS16		Device Reset Interrupt from any port
15-12	Reserved		Reserved
11	ICS11		Port3 Error
10	ICS10		Port2 Error
9	ICS9		Port1 Error
8	ICS8		Port0 Error
7-3	Reserved		Reserved
2	ICS2		Logical Layer Error Management Event Capture
1	ICS1		Port-write-in request received on any port
0	ICS0		Multi-cast event control symbol interrupt received on any port

5.25 Error, Reset, and Special Event Clear Interrupt Register (ERR_RST_EVNT_ICCR)

Figure 81. Error, Reset, and Special Event Clear Interrupt Register (ERR_RST_EVNT_ICCR)

31-17	16
Reserved	ICC16
R-0x00	W-0x00

LEGEND: R = Read only; -*n* = value after reset

15-12	11	10	9	8	7-3	2	1	0
Reserved	ICC11	ICC10	ICC9	ICC8	Reserved	ICC2	ICC1	ICC0
R-0x00	W-0x00	W-0x00	W-0x00	W-0x00	R-0x00	W-0x00	W-0x00	W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 55. Error, Reset, and Special Event Clear Interrupt Register (ERR_RST_EVNT_ICCR) Field Descriptions

Bit	Field	Value	Description
31-17	Reserved		Reserved
16	ICC16		Device Reset Interrupt from any port
15-12	Reserved		Reserved
11	ICC11		Port3 Error
10	ICC10		Port2 Error
9	ICC9		Port1 Error
8	ICC8		Port0 Error
7-3	Reserved		Reserved
2	ICC2		Logical Layer Error Management Event Capture
1	ICC1		Port-write-in request received on any port
0	ICC0		Multi-cast event control symbol interrupt received on any port

5.26 DOORBELLn Interrupt Condition Routing Register (DOORBELLn_ICRR)

Each of the four doorbells is supported by a register of this type.

Figure 82. DOORBELL*n* Interrupt Condition Routing Register (DOORBELL*n*_ICRR)

31 28	27 24	23 20	19 16
ICR7	ICR6	ICR5	ICR4
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

15 12	11 8	7 4	3 0
ICR3	ICR2	ICR1	ICR0
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

Table 56. DOORBELL*n* Interrupt Condition Routing Register (DOORBELL*n*_ICRR) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (0-7)		Doorbell *n* (0 to 3) CPU servicing interrupt condition routing bits

5.27 DOORBELLn Interrupt Condition Routing Register 2 (DOORBELLn_ICRR2)

Each of the four doorbells is supported by a register of this type.

Figure 83. DOORBELL*n* Interrupt Condition Routing Register 2 (DOORBELL*n*_ICRR2)

31 28	27 24	23 20	19 16
ICR15	ICR14	ICR13	ICR12
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00
15 12	**11 8**	**7 4**	**3 0**
ICR11	ICR10	ICR9	ICR8
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

Table 57. DOORBELL*n* Interrupt Condition Routing Register 2 (DOORBELL*n*_ICRR2) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (8-15)		Doorbell *n* (0 to 3) CPU servicing interrupt condition routing bits

5.28 RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR)

Figure 84. RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR)

31 28	27 24	23 20	19 16
ICR7	ICR6	ICR5	ICR4
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

15 12	11 8	7 4	3 0
ICR3	ICR2	ICR1	ICR0
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

Table 58. RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (0-7)		RX CPPI Interrupt condition routing bits

5.29 RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR2)

Figure 85. RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR2)

31 28	27 24	23 20	19 16
ICR15	ICR14	ICR13	ICR12
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

15 12	11 8	7 4	3 0
ICR11	ICR10	ICR9	ICR8
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

31-16
ICR (0-7)
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ICR (0-7)
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 59. RX CPPI Interrupt Condition Routing Register (RX_CPPI _ICRR2) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (8-15)		RX CPPI Interrupt condition routing bits

5.30 TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR)

Figure 86. TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

Table 60. TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (0-7)		TX CPPI Interrupt condition routing bits

5.31 TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR2)

Figure 87. TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR2)

31–28	27–24	23–20	19–16
ICR15	ICR14	ICR13	ICR12
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

15–12	11–8	7–4	3–0
ICR11	ICR10	ICR9	ICR8
R/W-0x00	R/W-0x00	R/W-0x00	R/W-0x00

LEGEND: R = Read, W = Write, n = value at reset

Table 61. TX CPPI Interrupt Condition Routing Register (TX_CPPI _ICRR2) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (8-15)		TX CPPI Interrupt condition routing bits

5.32 LSU Module Interrupt Condition Routing Register 0 (LSU_ICRR0)

Figure 88. LSU Module Interrupt Condition Routing Register 0 (LSU_ICRR0)

31–28	27–24	23–20	19–16
ICR7	ICR6	ICR5	ICR4
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15–12	11–8	7–4	3–0
ICR3	ICR2	ICR1	ICR0
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Table 62. LSU Module Interrupt Condition Routing Register 0 (LSU_ICRR0) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (7-0)		Load/Store module interrupt condition routing bits

5.33 LSU Module Interrupt Condition Routing Register 1 (LSU_ICRR1)

Figure 89. LSU Module Interrupt Condition Routing Register 1 (LSU_ICRR1)

31 28	27 24	23 20	19 16
ICR15	ICR14	ICR13	ICR12
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15 12	11 8	7 4	3 0
ICR11	ICR10	ICR9	ICR8
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Table 63. LSU Module Interrupt Condition Routing Register 1 (LSU_ICRR1) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (15-8)		Load/Store module interrupt condition routing bits

5.34 LSU Module Interrupt Condition Routing Register 2 (LSU_ICRR2)

Figure 90. LSU Module Interrupt Condition Routing Register 2 (LSU_ICRR2)

31 28	27 24	23 20	19 16
ICR23	ICR22	ICR21	ICR20
R/W-0000	R/W-0000	R/W-0000	R/W-0000

15 12	11 8	7 4	3 0
ICR19	ICR18	ICR17	ICR16
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Table 64. LSU Module Interrupt Condition Routing Register 2 (LSU_ICRR2) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (23-16)		Load/Store module interrupt condition routing bits

5.35 LSU Module Interrupt Condition Routing Register 3 (LSU_ICRR3)

Figure 91. LSU Module Interrupt Condition Routing Register 3 (LSU_ICRR3)

31 28	27 24	23 20	19 16
ICR31	ICR30	ICR29	ICR28
R/W-0000	R/W-0000	R/W-0000	R/W-0000
15 12	**11 8**	**7 4**	**3 0**
ICR27	ICR26	ICR25	ICR24
R/W-0000	R/W-0000	R/W-0000	R/W-0000

LEGEND: R = Read, W = Write, n = value at reset

Table 65. LSU Module Interrupt Condition Routing Register 3 (LSU_ICRR3) Field Descriptions

Bit	Field	Value	Description
31-0	ICR (31-24)		Load/Store module interrupt condition routing bits

5.36 Error, Reset, and Special Event Interrupt Condition Routing Register (ERR_RST_EVNT_ICRR)

Figure 92. Error, Reset, and Special Event Interrupt Condition Routing Register (ERR_RST_EVNT_ICRR)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
Reserved	ICR2	ICR1	ICR0
R-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 66. Error, Reset, and Special Event Interrupt Condition Routing Register (ERR_RST_EVNT_ICRR) Field Descriptions

Bit	Field	Value	Description
31-12	Reserved		Reserved
11-8	ICR2		Logical Layer Error Management Event Capture routing
7-4	ICR1		Routing of Port-write-in request received on any port
3-0	ICR0		Routing of Multi-cast event control symbol interrupt received on any port

5.37 Error, Reset, and Special Event Interrupt Condition Routing Register 2 (ERR_RST_EVNT_ICRR2)

Figure 93. Error, Reset, and Special Event Interrupt Condition Routing Register 2 (ERR_RST_EVNT_ICRR2)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
ICR11	ICR10	ICR9	ICR8
RW-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 67. Error, Reset, and Special Event Interrupt Condition Routing Register 2 (ERR_RST_EVNT_ICRR2) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-12	ICR11		Port3 Error routing
11-8	ICR10		Port2 Error routing
7-4	ICR9		Port1 Error routing
3-0	ICR8		Port0 Error routing

5.38 Error, Reset, and Special Event Interrupt Condition Routing Register 3 (ERR_RST_EVNT_ICRR3)

Figure 94. Error, Reset, and Special Event Interrupt Condition Routing Register 3 (ERR_RST_EVNT_ICRR3)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-4	3-0
Reserved	ICR16
R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 68. Error, Reset, and Special Event Interrupt Condition Routing Register 3 (ERR_RST_EVNT_ICRR3) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved		Reserved
3-0	ICR16		Routing of Device Reset Interrupt from any port

5.39 INTDSTn Interrupt Status Decode Registers (INTDSTn_DECODE)

There are eight of these registers.

Figure 95. INTDST*n* Interrupt Status Decode Registers (INTDST*n*_DECODE)

31-16
ISDR[31-16]
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ISDR[15-0]
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 69. INTDST*n* Interrupt Status Decode Registers (INTDST*n*_DECODE) Field Descriptions

Bit	Field	Value	Description
31-0	ISDR*n*		Interrupt sources that select a particular physical interrupt destination, are mapped to specific bits in the decode register. The interrupt sources are mapped to an interrupt decode register, only if the ICRR routes the interrupt source to the corresponding physical interrupt. The status decode bit is a logical "OR" of multiple interrupt sources that are mapped to the same bit.

5.40 INTDSTn Interrupt Rate Control Registers (INTDSTn_RATE_CNTL)

There are eight of these registers.

Figure 96. INTDST*n* Interrupt Rate Control Registers (INTDST*n*_RATE_CNTL)

31-16
COUNT_DOWN_VALUE
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
COUNT_DOWN_VALUE
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 70. INTDST*n* Interrupt Rate Control Registers (INTDST*n*_RATE_CNTL) Field Descriptions

Bit	Field	Value	Description
31-0	COUNT_DOWN_VALUE		The rate at which an interrupt can be generated is controllable for each physical interrupt destination. Rate control is implemented with a programmable down-counter. The counter reloads and immediately starts down-counting each time the CPU writes these registers. Once the rate control counter register is written, and the counter value reaches zero (note the CPU may write zero immediately for a zero count), the interrupt pulse generation logic is allowed to fire a single pulse if any bits in the corresponding ICSR register bits are set (or become set after the zero count is reached).

5.41 LSUn Control Register 0 (LSUn_REG0)

There are four of these registers, one for each LSU.

Figure 97. LSU*n* Control Register 0 (LSU*n*_REG0)

31-16
ADDRESS_MSB
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ADDRESS_MSB
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 71. LSU*n* Control Register 0 (LSU*n*_REG0) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_MSB		32 bit Ext address fields

5.42 LSUn Control Register 1 (LSUn_REG1)

There are four of these registers, one for each LSU.

Figure 98. LSU*n* Control Register 1 (LSU*n*_REG1)

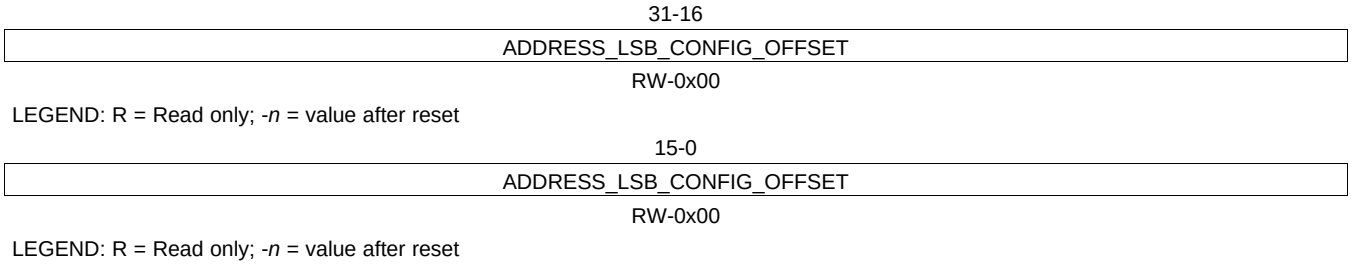


Table 72. LSU*n* Control Register 1 (LSU*n*_REG1) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_LSB_CONFIG_OFFSET		1. 32b Address- Packet Types 2,5, and 6 (will be used in conjunction with BYTE_COUNT to create 64b aligned RapidIO packet header address). 2. 24b Config-offset Field - Maintenance Packets Type 8 (will be used in conjunction with BYTE_COUNT to create 64b aligned RapidIO packet header Config_offset). The 2 lsb of this field must be zero since the smallest configuration access is 4B.

5.43 LSUn Control Register 2 (LSUn_REG2)

There are four of these registers, one for each LSU.

Figure 99. LSU*n* Control Register 2 (LSU*n*_REG2)

31-16
DSP_ADDRESS
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
DSP_ADDRESS
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 73. LSU*n* Control Register 2 (LSU*n*_REG2) Field Descriptions

Bit	Field	Value	Description
31-0	DSP_ADDRESS		32b DSP byte address

5.44 LSUn Control Register 3 (LSUn_REG3)

There are four of these registers, one for each LSU.

Figure 100. LSU*n* Control Register 3 (LSU*n*_REG3)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-12	11-0
Reserved	BYTE_COUNT
R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 74. LSU*n* Control Register 3 (LSU*n*_REG3) Field Descriptions

Bit	Field	Value	Description
31-12	Reserved		Reserved
11-0	BYTE_COUNT		Number of data bytes to Read/Write up to 4KB. (Used in conjunction with RapidIO address to create WRSIZE/RDSIZE and WDPTR in RapidIO packet header).

5.45 LSUn Control Register 4 (LSUn_REG4)

There are four of these registers, one for each LSU.

Figure 101. LSU*n* Control Register 4 (LSU*n*_REG4)

31-30	29-28	27-26	25-24	23-16
OUTPORTID	PRIORITY	XAMBS	ID_SIZE	DESTID
RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-1	0
DESTID	Reserved	INTERRUPT_REQ
RW-0x00	R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 75. LSU*n* Control Register 4 (LSU*n*_REG4) Field Descriptions

Bit	Field	Value	Description
31-30	OUTPORTID		Not applicable for Rapid IO header. Indicates the output port number for the packet to be transmitted from. Specified by the CPU along with NodeID.
29-28	PRIORITY		RapidIO prio field specifying packet priority. Request packets should not be sent at a priority level of 3 in order to avoid system deadlock. It is the responsibility of the software to assign the appropriate outgoing priority.
27-26	XAMBS		RapidIO xambs field specifying extended address MSB
25-24	ID_SIZE		RapidIO tt field specifying 8 or 16bit DeviceIDs
		00b	8 bit device Ids
		01b	16 bit device Ids
23-8	DESTID		RapidIO destinationID field specifying target device
7-1	Reserved		Reserved
0	INTERRUPT_REQ		CPU controlled request bit used for interrupt generation. Typically used in conjunction with Non-posted commands to alert the CPU when the requested data/status is present.
		0b	Interrupt is not requested upon completion of command
		1b	Interrupt is requested upon completion of command

5.46 LSUn Control Register 5 (LSUn_REG5)

There are four of these registers, one for each LSU.

Figure 102. LSU*n* Control Register 5 (LSU*n*_REG5)

31-16
DRBLL_INFO
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
HOP_COUNT	PACKET_TYPE
RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 76. LSU*n* Control Register 5 (LSU*n*_REG5) Field Descriptions

Bit	Field	Value	Description
31-16	DRBLL_INFO		RapidIO doorbell info field for type 10 packets
15-8	HOP_COUNT		RapidIO HOP_COUNT field specified for Type 8 Maintenance packets
7-0	PACKET_TYPE		4 msb = 4b ftype field for all packets and 4 lsb = 4b trans field for Packet types 2,5,8

5.47 LSUn Control Register 6 (LSUn_REG6)

There are four of these registers, one for each LSU.

Figure 103. LSU*n* Control Register 6 (LSU*n*_REG6)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-5	4-1	0
Reserved	COMPLETION_CODE	BSY
R-0x00	R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 77. LSU*n* Control Register 6 (LSU*n*_REG6) Field Descriptions

Bit	Field	Value	Description
31-5	Reserved		Reserved
4-1	COMPLETION_CODE		Indicates the status of the pending command.
		000b	Transaction complete, No Errors (Posted/Non-posted)
		001b	Transaction Timeout occurred on Non-posted transaction
		010 b	Transaction complete, Packet not sent due to flow control blockade (Xoff)
		011b	Transaction complete, Non-posted response packet (type 8 and 13) contained ERROR status, or response payload length was in error
		100b	Transaction complete, Packet not sent due to unsupported transaction type or invalid programming encoding for one or more LSU register fields
		101b	DMA data transfer error
		110b	"Retry" DOORBELL response received, or Atomic Test-and-swap was not allowed (semaphore in use)
		111b	Transaction complete, Packet not sent due to unavailable outbound credit at given priority
0	BSY		Indicates status of the command registers.
		0b	Command registers are available(writable) for next set of transfer descriptors
		1b	Command registers are busy with current transfer

5.48 LSU Congestion Control Flow Mask n (LSU_FLOW_MASKS n)

Figure 104. LSU Congestion Control Flow Mask *n* (LSU_FLOW_MASKS *n*)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
FLOW_MASK (0-15)
RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 78. LSU Congestion Control Flow Mask *n* (LSU_FLOW_MASKS *n*) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	FLOW_MASK (0-15)		LSU flow masks

5.49 Queue Transmit DMA Head Descriptor Pointer Registers (QUEUEn_TXDMA_HDP)

There are sixteen of these registers.

Figure 105. Queue Transmit DMA Head Descriptor Pointer Registers (QUEUE*n*_TXDMA_HDP)

31-16
TX_HDP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
TX_HDP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 79. Queue Transmit DMA Head Descriptor Pointer Registers (QUEUE*n*_TXDMA_HDP) Field Descriptions

Bit	Field	Value	Description
31-0	TX_HDP		This field is the host memory address for the first buffer descriptor in the transmit queue. This field is written by the host to initiate queue transmit operations and is zeroed by the port when all packets in the queue have been transmitted. An error condition results if the host writes this field when the current field value is nonzero. The address must be 32-bit word aligned .

5.50 Queue Transmit DMA Completion Pointer Registers (QUEUEn_TXDMA_CP)

There are sixteen of these registers.

Figure 106. Queue Transmit DMA Completion Pointer Registers (QUEUE*n*_TXDMA_CP)

31-16
TX_CP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
TX_CP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 80. Queue Transmit DMA Completion Pointer Registers (QUEUE*n*_TXDMA_CP) Field Descriptions

Bit	Field	Value	Description
31-0	TX_CP		This field is the host memory address for the transmit queue completion pointer. This register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The port uses the value written to determine if the interrupt should be deasserted.

5.51 Queue Receive DMA Head Descriptor Pointer Registers (QUEUEn_RXDMA_HDP)

There are sixteen of these registers.

Figure 107. Queue Receive DMA Head Descriptor Pointer Registers (QUEUE*n*_RXDMA_HDP)

31-16
RX_HDP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
RX_HDP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 81. Queue Receive DMA Head Descriptor Pointer Registers (QUEUE*n*_RXDMA_HDP) Field Descriptions

Bit	Field	Value	Description
31-0	RX_HDP		Rx Queue Head Descriptor Pointer: This field is the host memory address for the first buffer descriptor in the channel receive queue. This field is written by the host to initiate queue receive operations and is zeroed by the port when all free buffers have been used. An error condition results if the host writes this field when the current field value is nonzero. The address must be 32-bit word aligned.

5.52 Queue Receive DMA Completion Pointer Registers (QUEUEn_RXDMA_CP)

There are sixteen of these registers.

Figure 108. Queue Receive DMA Completion Pointer Registers (QUEUE*n*_RXDMA_CP)

31-16
RX_CP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
RX_CP
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 82. Queue Receive DMA Completion Pointer Registers (QUEUE*n*_RXDMA_CP) Field Descriptions

Bit	Field	Value	Description
31-0	RX_CP		Rx Queue Completion Pointer: This field is the host memory address for the receive queue completion pointer. This register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The port uses the value written to determine if the interrupt should be deasserted.

5.53 Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN)

Figure 109. Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUEUE15_TEAR_DWN	QUEUE14_TEAR_DWN	QUEUE13_TEAR_DWN	QUEUE12_TEAR_DWN	QUEUE11_TEAR_DWN	QUEUE10_TEAR_DWN	QUEUE9_TEAR_DWN	QUEUE8_TEAR_DWN	QUEUE7_TEAR_DWN	QUEUE6_TEAR_DWN	QUEUE5_TEAR_DWN	QUEUE4_TEAR_DWN	QUEUE3_TEAR_DWN	QUEUE2_TEAR_DWN	QUEUE1_TEAR_DWN	QUEUE0_TEAR_DWN
W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 83. Transmit Queue Teardown Register (TX_QUEUE_TEAR_DOWN) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15	QUEUE15_TEAR_DWN		Write = 1, tear down of Queue 15
14	QUEUE14_TEAR_DWN		Write = 1, tear down of Queue 14
13	QUEUE13_TEAR_DWN		Write = 1, tear down of Queue 13
12	QUEUE12_TEAR_DWN		Write = 1, tear down of Queue 12
11	QUEUE11_TEAR_DWN		Write = 1, tear down of Queue 11
10	QUEUE10_TEAR_DWN		Write = 1, tear down of Queue 10
9	QUEUE9_TEAR_DWN		Write = 1, tear down of Queue 9
8	QUEUE8_TEAR_DWN		Write = 1, tear down of Queue 8
7	QUEUE7_TEAR_DWN		Write = 1, tear down of Queue 7
6	QUEUE6_TEAR_DWN		Write = 1, tear down of Queue 6
5	QUEUE5_TEAR_DWN		Write = 1, tear down of Queue 5
4	QUEUE4_TEAR_DWN		Write = 1, tear down of Queue 4
3	QUEUE3_TEAR_DWN		Write = 1, tear down of Queue 3
2	QUEUE2_TEAR_DWN		Write = 1, tear down of Queue 2
1	QUEUE1_TEAR_DWN		Write = 1, tear down of Queue 1
0	QUEUE0_TEAR_DWN		Write = 1, tear down of Queue 0

5.54 Transmit CPPI Supported Flow Mask Registers n (TX_CPPI_FLOW_MASKSn)

There are eight registers of this type. See Figure 28 for more information on this register.

Figure 110. Transmit CPPI Supported Flow Mask Registers *n* (TX_CPPI_FLOW_MASKS*n*)

Transmit CPPI Supported Flow Mask Register 0 (TX_CPPI_FLOW_MASKS0)

31-16	15-0
QUEUE1_FLOW_MASK	QUEUE0_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 1 (TX_CPPI_FLOW_MASKS1)

31-16	15-0
QUEUE3_FLOW_MASK	QUEUE2_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 2 (TX_CPPI_FLOW_MASKS2)

31-16	15-0
QUEUE5_FLOW_MASK	QUEUE4_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 3 (TX_CPPI_FLOW_MASKS3)

31-16	15-0
QUEUE7_FLOW_MASK	QUEUE6_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 4 (TX_CPPI_FLOW_MASKS4)

31-16	15-0
QUEUE9_FLOW_MASK	QUEUE8_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 5 (TX_CPPI_FLOW_MASKS5)

31-16	15-0
QUEUE11_FLOW_MASK	QUEUE10_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 6 (TX_CPPI_FLOW_MASKS6)

31-16	15-0
QUEUE13_FLOW_MASK	QUEUE12_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Transmit CPPI Supported Flow Mask Register 7 (TX_CPPI_FLOW_MASKS7)

31-16	15-0
QUEUE15_FLOW_MASK	QUEUE14_FLOW_MASK
RW-0x01	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 84. Transmit CPPI Supported Flow Mask Registers *n* (TX_CPPI_FLOW_MASKS*n*) Field Descriptions

Field	Value	Description
QUEUE*n*_FLOW_MASK		Flow mask queue *n*
	0b	Transmit source does not support flow *n* from table entry for QUEUE*n*
	1b	Transmit source does support flow *n* from table entry for QUEUE*n*

5.55 Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN)

Figure 111. Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUEUE15_TEAR_DWN	QUEUE14_TEAR_DWN	QUEUE13_TEAR_DWN	QUEUE12_TEAR_DWN	QUEUE11_TEAR_DWN	QUEUE10_TEAR_DWN	QUEUE9_TEAR_DWN	QUEUE8_TEAR_DWN	QUEUE7_TEAR_DWN	QUEUE6_TEAR_DWN	QUEUE5_TEAR_DWN	QUEUE4_TEAR_DWN	QUEUE3_TEAR_DWN	QUEUE2_TEAR_DWN	QUEUE1_TEAR_DWN	QUEUE0_TEAR_DWN
W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00	W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 85. Receive Queue Teardown Register (RX_QUEUE_TEAR_DOWN) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15	QUEUE15_TEAR_DWN		Write = 1, tear down of Queue 15
14	QUEUE14_TEAR_DWN		Write = 1, tear down of Queue 14
13	QUEUE13_TEAR_DWN		Write = 1, tear down of Queue 13
12	QUEUE12_TEAR_DWN		Write = 1, tear down of Queue 12
11	QUEUE11_TEAR_DWN		Write = 1, tear down of Queue 11
10	QUEUE10_TEAR_DWN		Write = 1, tear down of Queue 10
9	QUEUE9_TEAR_DWN		Write = 1, tear down of Queue 9
8	QUEUE8_TEAR_DWN		Write = 1, tear down of Queue 8
7	QUEUE7_TEAR_DWN		Write = 1, tear down of Queue 7
6	QUEUE6_TEAR_DWN		Write = 1, tear down of Queue 6
5	QUEUE5_TEAR_DWN		Write = 1, tear down of Queue 5
4	QUEUE4_TEAR_DWN		Write = 1, tear down of Queue 4
3	QUEUE3_TEAR_DWN		Write = 1, tear down of Queue 3
2	QUEUE2_TEAR_DWN		Write = 1, tear down of Queue 2
1	QUEUE1_TEAR_DWN		Write = 1, tear down of Queue 1
0	QUEUE0_TEAR_DWN		Write = 1, tear down of Queue 0

5.56 Receive CPPI Control Register (RX_CPPI_CNTL)

Figure 112. Receive CPPI Control Register (RX_CPPI_CNTL)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
QUEUE15_IN_ORDER	QUEUE14_IN_ORDER	QUEUE13_IN_ORDER	QUEUE12_IN_ORDER	QUEUE11_IN_ORDER	QUEUE10_IN_ORDER	QUEUE9_IN_ORDER	QUEUE8_IN_ORDER	QUEUE7_IN_ORDER	QUEUE6_IN_ORDER	QUEUE5_IN_ORDER	QUEUE4_IN_ORDER	QUEUE3_IN_ORDER	QUEUE2_IN_ORDER	QUEUE1_IN_ORDER	QUEUE0_IN_ORDER
RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 86. Receive CPPI Control Register (RX_CPPI_CNTL) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	QUEUE*n*_IN_ORDER (0-15)		Queue*n* In Order (QUEUE(0-15)_IN_ORDER)
		0b	Allows out-of-order message
		1b	Only allows in-order messages. Used for applications with dedicated source-destination flows.

5.57 Transmit CPPI Weighted Round Robin Control Register 0 (TX_QUEUE_CNTL0)

Figure 113. Transmit CPPI Weighted Round Robin Control Register 0 (TX_QUEUE_CNTL0)

31-28	27-24	23-20	19-16
TX_QUEUE_MAP3_NUM_MSGS	TX_QUEUE_MAP3_QUEUE_PTR	TX_QUEUE_MAP2_NUM_MSGS	TX_QUEUE_MAP2_QUEUE_PTR
RW-0x00	RW-0x03	RW-0x00	RW-0x02

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
TX_QUEUE_MAP1_NUM_MSGS	TX_QUEUE_MAP1_QUEUE_PTR	TX_QUEUE_MAP0_NUM_MSGS	TX_QUEUE_MAP0_QUEUE_PTR
RW-0x00	RW-0x01	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 87. Transmit CPPI Weighted Round Robin Control Register 0 (TX_QUEUE_CNTL0) Field Descriptions

Bit	Field	Value	Description
31-28	TX_QUEUE_MAP3_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map4
27-24	TX_QUEUE_MAP3_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
23-20	TX_QUEUE_MAP2_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map3
19-16	TX_QUEUE_MAP2_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
15-12	TX_QUEUE_MAP1_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map2
11-8	TX_QUEUE_MAP1_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
7-4	TX_QUEUE_MAP0_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map1
3-0	TX_QUEUE_MAP0_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues

5.58 Transmit CPPI Weighted Round Robin Control Register 1 (TX_QUEUE_CNTL1)

Figure 114. Transmit CPPI Weighted Round Robin Control Register 1 (TX_QUEUE_CNTL1)

31-28	27-24	23-20	19-16
TX_QUEUE_MAP7_NUM_MSGS	TX_QUEUE_MAP7_QUEUE_PTR	TX_QUEUE_MAP6_NUM_MSGS	TX_QUEUE_MAP6_QUEUE_PTR
RW-0x00	RW-0x07	RW-0x00	RW-0x06

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
TX_QUEUE_MAP5_NUM_MSGS	TX_QUEUE_MAP5_QUEUE_PTR	TX_QUEUE_MAP4_NUM_MSGS	TX_QUEUE_MAP4_QUEUE_PTR
RW-0x00	RW-0x05	RW-0x00	RW-0x04

LEGEND: R = Read only; -*n* = value after reset

Table 88. Transmit CPPI Weighted Round Robin Control Register 1 (TX_QUEUE_CNTL1) Field Descriptions

Bit	Field	Value	Description
31-28	TX_QUEUE_MAP7_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map8
27-24	TX_QUEUE_MAP7_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
23-20	TX_QUEUE_MAP6_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map7
19-16	TX_QUEUE_MAP6_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
15-12	TX_QUEUE_MAP5_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map6
11-8	TX_QUEUE_MAP5_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
7-4	TX_QUEUE_MAP4_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map5
3-0	TX_QUEUE_MAP4_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues

5.59 Transmit CPPI Weighted Round Robin Control Register 2 (TX_QUEUE_CNTL2)

Figure 115. Transmit CPPI Weighted Round Robin Control Register 2 (TX_QUEUE_CNTL2)

31-28	27-24	23-20	19-16
TX_QUEUE_MAP11_NUM_MSGS	TX_QUEUE_MAP11_QUEUE_PTR	TX_QUEUE_MAP10_NUM_MSGS	TX_QUEUE_MAP10_QUEUE_PTR
RW-0x00	RW-0x0B	RW-0x00	RW-0x0A

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
TX_QUEUE_MAP9_NUM_MSGS	TX_QUEUE_MAP9_QUEUE_PTR	TX_QUEUE_MAP8_NUM_MSGS	TX_QUEUE_MAP8_QUEUE_PTR
RW-0x00	RW-0x09	RW-0x00	RW-0x08

LEGEND: R = Read only; -*n* = value after reset

Table 89. Transmit CPPI Weighted Round Robin Control Register 2 (TX_QUEUE_CNTL2) Field Descriptions

Bit	Field	Value	Description
31-28	TX_QUEUE_MAP11_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map12
27-24	TX_QUEUE_MAP11_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
23-20	TX_QUEUE_MAP10_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map11
19-16	TX_QUEUE_MAP10_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
15-12	TX_QUEUE_MAP9_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map10
11-8	TX_QUEUE_MAP9_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
7-4	TX_QUEUE_MAP8_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map9
3-0	TX_QUEUE_MAP8_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues

5.60 Transmit CPPI Weighted Round Robin Control Register 3 (TX_QUEUE_CNTL3)

Figure 116. Transmit CPPI Weighted Round Robin Control Register 3 (TX_QUEUE_CNTL3)

31-28	27-24	23-20	19-16
TX_QUEUE_MAP15_NUM_MSGS	TX_QUEUE_MAP15_QUEUE_PTR	TX_QUEUE_MAP14_NUM_MSGS	TX_QUEUE_MAP14_QUEUE_PTR
RW-0x00	RW-0x0F	RW-0x00	RW-0x0E

LEGEND: R = Read only; -*n* = value after reset

15-12	11-8	7-4	3-0
TX_QUEUE_MAP13_NUM_MSGS	TX_QUEUE_MAP13_QUEUE_PTR	TX_QUEUE_MAP12_NUM_MSGS	TX_QUEUE_MAP12_QUEUE_PTR
RW-0x00	RW-0x0D	RW-0x00	RW-0x0C

LEGEND: R = Read only; -*n* = value after reset

Table 90. Transmit CPPI Weighted Round Robin Control Register 3 (TX_QUEUE_CNTL3) Field Descriptions

Bit	Field	Value	Description
31-28	TX_QUEUE_MAP15_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map0
27-24	TX_QUEUE_MAP15_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
23-20	TX_QUEUE_MAP14_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map15
19-16	TX_QUEUE_MAP14_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
15-12	TX_QUEUE_MAP13_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map14
11-8	TX_QUEUE_MAP13_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues
7-4	TX_QUEUE_MAP12_NUM_MSGS		Number of contiguous messages (descriptors) to process before moving to TX_Queue_Map13
3-0	TX_QUEUE_MAP12_QUEUE_PTR		Pointer to a Queue, programmable to any of the 16 TX queues

5.61 Mailbox-to-Queue Mapping Register Ln (RXU_MAP_Ln)

Figure 117. Mailbox-to-Queue Mapping Register L*n* (RXU_MAP_L*n*)

31-30	29-24	23-22	21-16
LETTER_MASK	MAILBOX_MASK	LETTER	MAILBOX
RW-0x03	RW-0x3F	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
SOURCEID
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 91. Mailbox-to-Queue Mapping Register L*n* (RXU_MAP_L*n*) Field Descriptions

Bit	Field	Value	Description
31-30	LETTER_MASK		Allowed letter mask for this mapping register
29-24	MAILBOX_MASK		Allowed mail box mask for this mapping register
23-22	LETTER		Allowed letter for this mapping register
21-16	MAILBOX		Allowed mail box for this mapping register
15-0	SOURCEID		The SOURCEID field is used to indicate which external device has access to the mapping entry and corresponding queue. A compare is performed between the sourceID of the incoming message packet and each relevant mailbox/letter table mapping entry SOURCEID field. A miscompare results in an ERROR response back to the sender and the transaction is logged in the Logical Layer Error Management capture registers.

5.62 Mailbox-to-Queue Mapping Register Hn (RXU_MAP_Hn)

Figure 118. Mailbox-to-Queue Mapping Register H*n* (RXU_MAP_H*n*)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-10	9-8	7-6	5-2	1	0
Reserved	TT	Reserved	QUEUE_ID	PROMISCUOUS	SEGMENT_MAPPING
R-0x00	RW-0x01	R-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 92. Mailbox-to-Queue Mapping Register H*n* (RXU_MAP_H*n*) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved		Reserved
9-8	TT		Transport type
		0b	Matches the 8 LSB of the source ID
		1b	Matches the full 16 bits of the source ID
7-6	Reserved		Reserved
5-2	QUEUE_ID		Corresponding queue 0 to queue 15
1	PROMISCUOUS		Promiscuous =1, full access to the queue for any source ID
		0b	Promiscuous bit enabled
		1b	Promiscuous bit disabled
0	SEGMENT_MAPPING		Segment mapping
		0b	Single segment: all six bits of mailbox valid
		1b	Multi segment: only 2 LSBs of mailbox valid

5.63 Flow Control Table Entry Registers (FLOW_CNTLn)

There are sixteen of these registers.

Figure 119. Flow Control Table Entry Registers (FLOW_CNTL*n*)

31-18	17-16
Reserved	TT
R-0x00	RW-0x01

LEGEND: R = Read only; -*n* = value after reset

15-0
FLOW_CNTL_ID
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 93. Flow Control Table Entry Registers (FLOW_CNTL*n*) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved		Reserved
17-16	TT		Selects Flow-Cntl-ID length
		00b	8 bit Ids
		01b	16 bit Ids
		10b-11b	Reserved
15-0	FLOW_CNTL_ID		DestID of Flow *n*

5.64 Device Identity CAR (DEV_ID)

Figure 120. Device Identity CAR (DEV_ID)

31-16

DEVICEIDENTITY

R-0x0000

LEGEND: R = Read only; -*n* = value after reset

15-0

DEVICE_VENDORIDENTITY

R-0x0030

LEGEND: R = Read only; -*n* = value after reset

Table 94. Device Identity CAR (DEV_ID) Field Descriptions

Bit	Field	Value	Description
31-16	DEVICEIDENTITY		Identifies the type of device. Vendor specific.
15-0	DEVICE_VENDORIDENTITY		Device Vendor ID assigned by RapidIO TA

5.65 Device Information CAR (DEV_INFO)

Figure 121. Device Information CAR (DEV_INFO)

31-16
DEVICEREV
R-0x0000

LEGEND: R = Read only; -*n* = value after reset

15-0
DEVICEREV
R-0x0000

LEGEND: R = Read only; -*n* = value after reset

Table 95. Device Information CAR (DEV_INFO) Field Descriptions

Bit	Field	Value	Description
31-0	DEVICEREV		Vendor supply device revision

5.66 Assembly Identity CAR (ASBLY_ID)

Figure 122. Assembly Identity CAR (ASBLY_ID)

31-16
ASSY_IDENTITY
R-0x0000

LEGEND: R = Read only; -*n* = value after reset

15-0
ASSY_VENDORIDENTITY
R-0x0030

LEGEND: R = Read only; -*n* = value after reset

Table 96. Assembly Identity CAR (ASBLY_ID) Field Descriptions

Bit	Field	Value	Description
31-16	ASSY_IDENTITY		Assembly Identifier. Vendor Specific.
15-0	ASSY_VENDORIDENTITY		Assembly Vendor Identifier assigned by RapidIO TA.

5.67 Assembly Information CAR (ASBLY_INFO)

Figure 123. Assembly Information CAR (ASBLY_INFO)

31-16
ASSYREV
R-0x0000

LEGEND: R = Read only; -*n* = value after reset

15-0
EXTENDEDFEATURESPTR
R-0x0100

LEGEND: R = Read only; -*n* = value after reset

Table 97. Assembly Information CAR (ASBLY_INFO) Field Descriptions

Bit	Field	Value	Description
31-16	ASSYREV		Assembly revision level
15-0	EXTENDEDFEATURESPTR		Pointer to first entry in extended features list

5.68 Processing Element Features CAR (PE_FEAT)

Figure 124. Processing Element Features CAR (PE_FEAT)

31	30	29	28	27-16
BRIDGE	MEMORY	PROCESSOR	SWITCH	Reserved
R-0x00	R-0x00	R-0x01	R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7	6	5	4	3	2-0
Reserved	FLOW_CONTROL_SUPPORT	RETRANSMIT_SUPPRESS	CRF_SUPPORT	LARGE_SUPPORT	EXTENDED_FEATURES	EXTENDED_ADDRESSING_SUPPORT
R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x01	R-0x0001

LEGEND: R = Read only; -*n* = value after reset

Table 98. Processing Element Features CAR (PE_FEAT) Field Descriptions

Bit	Field	Value	Description
31	BRIDGE		PE can bridge to another interface. Examples are PCI, proprietary processor buses, DRAM, etc.
30	MEMORY		PE has physically addressable local address space and can be accessed as an endpoint through non-maintenance (i.e., non-coherent read and write) operations. This local address space may be limited to local configuration Registers, or could be on-chip SRAM, etc.
29	PROCESSOR		PE physically contains a local processor or similar device that executes code. A device that bridges to an interface that connects to a processor does not count (see bit 31).
28	SWITCH		PE can bridge to another external RapidIO interface- an internal port to a local endpoint does not count as a switch port. For example, a device with two RapidIO ports and a local endpoint is a two port switch, not a three port switch, regardless of the internal architecture.
27-8	Reserved		Reserved
7	FLOW_CONTROL_SUPPORT		PE supports congestion flow control mechanism
		0b	PE does not support flow control
		1b	PE supports flow control
6	RETRANSMIT_SUPPRESS		PE supports suppression of error recovery on packet CRC errors.
		0b	PE does not support suppression
		1b	PE supports suppression
5	CRF_SUPPORT		This bit indicates PE support for the Critical Request Flow (CRF) Function.
		0b	PE does not support CRF Function
		1b	PE supports CRF Function
4	LARGE_SUPPORT		Support of common transport large systems.
		0b	PE does not support common transport large systems
		1b	PE supports common transport large systems
3	EXTENDED_FEATURES		PE has extended features list; the extended features pointer is valid.
2-0	EXTENDED_ADDRESSING_SUPPORT		Indicates the number address bits supported by the PE both as a source and target of an operation. All PEs shall at minimum support 34 bit addresses. Encodings other than below are reserved.
		111b	PE supports 66, 50 and 34 bit addresses
		101b	PE supports 66 and 34 bit addresses
		011b	PE supports 50 and 34 bit addresses
		001b	PE supports 34 bit addresses

5.69 Source Operations CAR (SRC_OP)

Figure 125. Source Operations CAR (SRC_OP)

31-18	17-16
Reserved	IMPLMNT_DEFINED_2
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1-0
READ	WRITE	STREAM_WRITE	WRITE_WITH_RESP	DATA_MESS	DOORBELL	Reserved	ATOMIC_TEST_AND_SWAP	ATOMIC_INCRMNT	ATOMIC_DCRMNT	ATOMIC_SET	ATOMIC_CLEAR	Reserved	PORT_WRITE	IMPLMNT_DEFINED_1
R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x01	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 99. Source Operations CAR (SRC_OP) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved		Reserved
17-16	IMPLMNT_DEFINED_2		Defined by the device implementation
15	READ		PE can support a read operation
14	WRITE		PE can support a write operation
13	STREAM_WRITE		PE can support a streaming-write operation
12	WRITE_WITH_RESP		PE can support a write-with-response operation
11	DATA_MESS		PE can support a data message operation
10	DOORBELL		PE can support a doorbell operation
9	Reserved		Reserved
8	ATOMIC_TEST_AND_SWAP		PE can support an atomic test-and-swap operation
7	ATOMIC_INCRMNT		PE can support an atomic increment operation
6	ATOMIC_DCRMNT		PE can support an atomic decrement operation
5	ATOMIC_SET		PE can support an atomic set operation
4	ATOMIC_CLEAR		PE can support an atomic clear operation
3	Reserved		Reserved
2	PORT_WRITE		PE can support a port-write generation
1-0	IMPLMNT_DEFINED_1		Defined by the device implementation

5.70 Destination Operations CAR (DEST_OP)

Figure 126. Destination Operations CAR (DEST_OP)

31-18	17-16
Reserved	IMPLMNT_DEFINED_2
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1-0
READ	WRITE	STREAM_WRITE	WRITE_WITH_RESP	DATA_MESS	DOORBELL	Reserved	ATOMIC_TEST_AND_SWAP	ATOMIC_INCRMNT	ATOMIC_DCRMNT	ATOMIC_SET	ATOMIC_CLEAR	Reserved	PORT_WRITE	IMPLMNT_DEFINED_1
R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x00	R-0x01	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 100. Destination Operations CAR (DEST_OP) Field Descriptions

Bit	Field	Value	Description
31-18	Reserved		Reserved
17-16	IMPLMNT_DEFINED_2		Defined by the device implementation
15	READ		PE can support a read operation
14	WRITE		PE can support a write operation
13	STREAM_WRITE		PE can support a streaming-write operation
12	WRITE_WITH_RESP		PE can support a write-with-response operation
11	DATA_MESS		PE can support a data message operation
10	DOORBELL		PE can support a doorbell operation
9	Reserved		Reserved
8	ATOMIC_TEST_AND_SWAP		PE can support an atomic test-and-swap operation
7	ATOMIC_INCRMNT		PE can support an atomic increment operation
6	ATOMIC_DCRMNT		PE can support an atomic decrement operation
5	ATOMIC_SET		PE can support an atomic set operation
4	ATOMIC_CLEAR		PE can support an atomic clear operation
3	Reserved		Reserved
2	PORT_WRITE		PE can support a port-write generation
1-0	IMPLMNT_DEFINED_1		Defined by the device implementation

5.71 Processing Element Logical Layer Control CSR (PE_LL_CTL)

Figure 127. Processing Element Logical Layer Control CSR (PE_LL_CTL)

31-16
Reserved
R-0x0000

LEGEND: R = Read only; -*n* = value after reset

15-3	2-0
Reserved	EXTENDED_ADDRESSING_CONTROL
R-0x0000	RW-0x0001

LEGEND: R = Read only; -*n* = value after reset

Table 101. Processing Element Logical Layer Control CSR (PE_LL_CTL) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved		Reserved
2-0	EXTENDED_ADDRESSING_CONTROL		Controls the number of address bits generated by the PE as a source and processed by the PE as the target of an operation. All other encodings reserved.
		100b	PE supports 66 bit addresses
		010b	PE supports 50 bit addresses
		001b	PE supports 34 bit addresses

5.72 Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR)

Figure 128. Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR)

31	30-16
Reserved	LCSBA
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
LCSBA
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 102. Local Configuration Space Base Address 0 CSR (LCL_CFG_HBAR) Field Descriptions

Bit	Field	Value	Description
31	Reserved		Reserved
30-0	LCSBA		For bits 30 to 15: Reserved for 34b addresses, reserved for 50b addresses, bits 66-51 of a 66b address. For bits 14 to 0: Reserved for 34b addresses, bits 50-36 of a 50b address, bits 50-36 of a 66b address.

5.73 Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR)

Figure 129. Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR)

31-16
LCSBA
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
LCSBA
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 103. Local Configuration Space Base Address 1 CSR (LCL_CFG_BAR) Field Descriptions

Bit	Field	Value	Description
31-0	LCSBA		For bit 31: Reserved for 34b addresses, bit 35 of a 50b address, bit 35 of a 66b address. For bits 30 to 0: Bits 34-3 of a 34b address, bits 35-3 of a 50b address. bits 35-3 of a 66b address.

5.74 Base Device ID CSR (BASE_ID)

Figure 130. Base Device ID CSR (BASE_ID)

31-24	23-16
Reserved	BASE_DEVICEID
R-0x00	RW-0x00FF

LEGEND: R = Read only; -*n* = value after reset

15-0
LARGE_BASE_DEVICEID
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

Table 104. Base Device ID CSR (BASE_ID) Field Descriptions

Bit	Field	Value	Description
31-24	Reserved		Reserved
23-16	BASE_DEVICEID		This is the base ID of the device in small common transport system (endpoints only).
15-0	LARGE_BASE_DEVICEID		This is the base ID of the device in a large common transport system (Only valid for endpoints, and if bit 4 of the PE_FEAT Register is set).

5.75 Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK)

See Section 2.4.2 of the RapidIO Specification for description of this register. It provides a lock function that is write-once/reset-able.

Figure 131. Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
HOST_BASE_DEVICEID
RW-0xFFFF

LEGEND: R = Read only; -*n* = value after reset

Table 105. Host Base Device ID Lock CSR (HOST_BASE_ID_LOCK) Field Descriptions

Bit	Field	Value	Description
31-16	Reserved		Reserved
15-0	HOST_BASE_DEVICEID		This is the base ID for the Host PE that is initializing this PE.

5.76 Component Tag CSR (COMP_TAG)

Figure 132. Component Tag CSR (COMP_TAG)

31-16
COMPONENT_TAG
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
COMPONENT_TAG
RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 106. Component Tag CSR (COMP_TAG) Field Descriptions

Bit	Field	Value	Description
31-0	COMPONENT_TAG		Software defined component Tag for PE. Useful for devices without device IDs.

5.77 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD)

Figure 133. 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD)

31-16
EF_PTR
R-0x1000

LEGEND: R = Read only; -*n* = value after reset

15-0
EF_ID
R-0x0001

LEGEND: R = Read only; -*n* = value after reset

Table 107. 1x/4x LP_Serial Port Maintenance Block Header Register (SP_MB_HEAD) Field Descriptions

Bit	Field	Value	Description
31-16	EF_PTR		Hard wired pointer to the next block in the data structure.
15-0	EF_ID		Hard wired extended features ID.
		0x0001	General endpoint device
		0x0002	General endpoint device with software assisted error recovery option
		0x0003	Switch

5.78 Port Link Time-Out Control CSR (SP_LT_CTL)

Figure 134. Port Link Time-Out Control CSR (SP_LT_CTL)

31-16
TIMEOUT_VALUE
RW-0xFFFFFF

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
TIMEOUT_VALUE	Reserved
RW-0xFFFFFF	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 108. Port Link Timeout Control CSR (SP_LT_CTL) Field Descriptions

Bit	Field	Value	Description
31-8	TIMEOUT_VALUE		Timeout value for all ports on the device. This timeout is for link events such as sending a packet to receiving the corresponding ACK. Max value represents 3-6 seconds. Timeout duration = 205nS * Timeout Value; where Timeout value is the decimal representation of this register value.
		FFFFFFh	3.4 seconds
		0FFFFFh	215 ms
		00FFFFh	13.4 ms
		000FFFh	839.5 us
		0000FFh	52.3 us
		00000Fh	3.1 us
		000001h	205 ns for simulation only
		000000h	Timer disabled
7-0	Reserved		Reserved

5.79 Port Response Time-Out Control CSR (SP_RT_CTL)

Figure 135. Port Response Time-Out Control CSR (SP_RT_CTL)

31-16
TIMEOUT_VALUE
RW-0xFFFFFF

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
TIMEOUT_VALUE	Reserved
RW-0xFFFFFF	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 109. Port Response Time-Out Control CSR (SP_RT_CTL) Field Descriptions

Bit	Field	Value	Description
31-8	TIMEOUT_VALUE		Timeout value for all ports on the device. This timeout is for sending a packet to receiving the corresponding response packet. Max value represents 3-6 seconds. The timeout duration can be expressed as: Timeout = 15 * ((Prescale value + 1) * DMA clock period * Timeout Value); where Prescale value is set in 0x0020 PSCR and the Timeout value is the decimal representation of this register value. Example: 400Mhz DMA, Prescalar 0100b, Timeout Value FFFFFFh; Timeout duration = 15*(4+1)*2.5nS*16777216 = 3.15s.
7-0	Reserved		Reserved

5.80 Port General Control CSR (SP_GEN_CTL)

Figure 136. Port General Control CSR (SP_GEN_CTL)

31	30	29	28-16
HOST	MASTER_ENABLE	DISCOVERED	Reserved
RW-0x00	RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 110. Port General Control CSR (SP_GEN_CTL) Field Descriptions

Bit	Field	Value	Description
31	HOST		A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are typically initialized by Host devices.
		0b	Agent or Slave device
		1b	Host device
30	MASTER_ENABLE		The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests.
		0b	Processing element cannot issue requests
		1b	Processing element can issue requests
29	DISCOVERED		This device has been located by the processing element responsible for system configuration.
		0b	The device has not been previously discovered
		1b	The device has been discovered by another processing element
28-0	Reserved		Reserved

5.81 Port Link Maintenance Request CSR n (SPn_LM_REQ)

Each of the four ports is supported by a register of this type.

Figure 137. Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-3	2-0
Reserved	COMMAND
R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 111. Port Link Maintenance Request CSR *n* (SP*n*_LM_REQ) Field Descriptions

Bit	Field	Value	Description
31-3	Reserved		Reserved
2-0	COMMAND		A write to this register generates a link-request control symbol on the corresponding port interface. Command to be sent in the link-request control symbol. If read, this field returns the last written value

5.82 Port Link Maintenance Response CSR n (SPn_LM_RESP)

Each of the four ports is supported by a register of this type.

Figure 138. Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP)

31	30-16
RESPONSE_VALID	Reserved
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-10	9-5	4-0
Reserved	ACKID_STATUS	LINK_STATUS
R-0x00	R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 112. Port Link Maintenance Response CSR *n* (SP*n*_LM_RESP) Field Descriptions

Bit	Field	Value	Description
31	RESPONSE_VALID		If the link-request causes a link-response, this bit indicates that the link-response has been received and the status fields are valid. If the link-request does not cause a link-response, this bit indicates that the link-request has been transmitted. This bit automatically clears on read.
30-10	Reserved		Reserved
9-5	ACKID_STATUS		AckID status field from the link-response control symbol
4-0	LINK_STATUS		Link status field from the link-response control symbol

5.83 Port Local AckID Status CSR n (SPn_ACKID_STAT)

Each of the four ports is supported by a register of this type.

Figure 139. Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT)

31-29	28-24	23-16
Reserved	INBOUND_ACKID	Reserved
R-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-13	12-8	7-5	4-0
Reserved	OUTSTANDING_ACKID	Reserved	OUTBOUND_ACKID
R-0x00	RW-0x00	R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 113. Port Local AckID Status CSR *n* (SP*n*_ACKID_STAT) Field Descriptions

Bit	Field	Value	Description
31-29	Reserved		Reserved
28-24	INBOUND_ACKID		Input port next expected ackID value
23-13	Reserved		Reserved
12-8	OUTSTANDING_ACKID		Output port unacknowledged ackID status. Next expected acknowledge control symbol ackID field that indicates the ackID value expected in the next received acknowledge control symbol.
7-5	Reserved		Reserved
4-0	OUTBOUND_ACKID		Output port next transmitted ackID value. Software writing this value can force retransmission of outstanding unacknowledged packets in order to manually implement error recovery.

5.84 Port Error and Status CSR n (SPn_ERR_STAT)

Each of the four ports is supported by a register of this type.

Figure 140. Port Error and Status CSR *n* (SP*n*_ERR_STAT)

31-27	26	25	24	23-21	20	19	18	17	16
Reserved	OUTPUT_PKT_DROP	OUTPUT_FLD_ENC	OUTPUT_DEGRD_ENC	Reserved	OUTPUT_RETRY_ENC	OUTPUT_RETRIED	OUTPUT_RETRY_STP	OUTPUT_ERROR_ENC	OUTPUT_ERROR_STP
R-0x00	RC-0x00	RC-0x00	RC-0x00	R-0x00	RC-0x00	R-0x00	R-0x00	RC-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-11	10	9	8	7-5	4	3	2	1	0
Reserved	INPUT_RETRY_STP	INPUT_ERROR_ENC	INPUT_ERROR_STP	Reserved	PORT_WRITE_PND	Reserved	PORT_ERROR	PORT_OK	PORT_UNINITIALIZED
R-0x00	R-0x00	RC-0x00	R-0x00	R-0x00	RC-0x00	R-0x00	RC-0x00	R-0x00	R-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 114. Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions

Bit	Field	Value	Description
31-27	Reserved		Reserved
26	OUTPUT_PKT_DROP		Output port has discarded a packet. (switch devices only)
25	OUTPUT_FLD_ENC		Output port has encountered a failed condition, meaning that the failed port error threshold has been reached in the Port *n* Error Rate Threshold Register. Once set, it remains set until written with a logic 1 to clear.
24	OUTPUT_DEGRD_ENC		Output port has encountered a degraded condition, meaning that the degraded port error threshold has been reached in the Port *n* Error Rate Threshold Register. Once set, it remains set until written with a logic 1 to clear.
23-21	Reserved		Reserved
20	OUTPUT_RETRY_ENC		Output port has encountered a retry condition. This bit is set when bit 18 is set. Once set, remains set until written with a logic 1 to clear.
19	OUTPUT_RETRIED		Output port has received a packet-retry control symbol and can not make forward progress. This bit is set when bit 18 is set and is cleared when a packet-accepted or a packet-not-accepted control symbol is received (read-only).
18	OUTPUT_RETRY_STP		Output port has received a packet-retry control symbol and is in the "output retry-stopped" state (read-only).
17	OUTPUT_ERROR_ENC		Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 16 is set. Once set, remains set until written with a logic 1 to clear
16	OUTPUT_ERROR_STP		Output is in the "output error-stopped" state (read-only).
15-11	Reserved		Reserved
10	INPUT_RETRY_STP		Input port is in the "input retry-stopped" state (read-only).
9	INPUT_ERROR_ENC		Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 8 is set. Once set, remains set until written with a logic 1 to clear
8	INPUT_ERROR_STP		Input port is in the "input error-stopped" state (read-only).
7-5	Reserved		Reserved
4	PORT_WRITE_PND		Port has encountered a condition which required it to initiate a Maintenance Port-write operation This bit is only valid if the device is capable of issuing a maintenance port-write transaction. Once set remains set until written with a logic 1 to clear.
3	Reserved		Reserved
2	PORT_ERROR		Input or output port has encountered an error from which hardware was unable to recover. Once set, remains set until written with a logic 1 to clear.

Table 114. Port Error and Status CSR *n* (SP*n*_ERR_STAT) Field Descriptions (continued)

Bit	Field	Value	Description
1	PORT_OK		The input and output ports are initialized and the port is exchanging error-free control symbols with the attached device (read-only).
0	PORT_UNINITIALIZED		Input and output ports are not initialized. This bit and bit 1 are mutually exclusive (read-only).

5.85 Port Control CSR n (SPn_CTL)

Each of the four ports is supported by a register of this type.

Figure 141. Port Control CSR *n* (SP*n*_CTL)

31-30	29-27	26-24	23	22	21	20	19	18-16
PORT_WIDTH	INITIALIZED_PORT_WIDTH	PORT_WIDTH_OVERRIDE	PORT_DISABLE	OUTPUT_PORT_ENABLE	INPUT_PORT_ENABLE	ERROR_CHECK_DISABLE	MULTICAST_PARTICIPANT	Reserved
R-0x01	R-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-4	3	2	1	0
Reserved	STOP_PORT_FLD_ENC_ENABLE	DROP_PACKET_ENABLE	PORT_LOCKOUT	PORT_TYPE
R-0x00	RW-0x00	RW-0x00	RW-0x00	R-0x01

LEGEND: R = Read only; -*n* = value after reset

Table 115. Port Control CSR *n* (SP*n*_CTL) Field Descriptions

Bit	Field	Value	Description
31-30	PORT_WIDTH		Hardware width of the port (read only). Only 00b is valid for SP1, SP2, and SP3
		00b	Single-lane port
		01b	Four-lane port
		10b-11b	Reserved
29-27	INITIALIZED_PORT_WIDTH		Width of the ports after initialized (read only)
		000b	Single-lane port, lane 0
		001b	Single-lane port, lane 2
		010b	Four-lane port
		011b-111b	Reserved
26-24	PORT_WIDTH_OVERRIDE		Soft port configuration to override the hardware size
		000b	No override
		001b	Reserved
		010b	Force single lane, lane 0
		011b	Force single lane, lane 2
		100b-111b	Reserved
23	PORT_DISABLE		Port disable
		0b	Port receivers/drivers are enabled
		1b	Port receivers/drivers are disabled and are unable to receive/transmit to any packets or control symbols
22	OUTPUT_PORT_ENABLE		Output port enable
		0b	Port is stopped and not enabled to issue any packets except to route or respond to I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Control symbols are not affected and are sent normally.
		1b	Port is enabled to issue any packets

Table 115. Port Control CSR *n* (SP*n*_CTL) Field Descriptions (continued)

Bit	Field	Value	Description
21	INPUT_PORT_ENABLE		Input port receive enable
		0b	Port is stopped and only enabled to route or respond I/O logical MAINTENANCE packets, depending upon the functionality of the processing element. Other packets generate packet-not-accepted control symbols to force an error condition to be signaled by the sending device. Control symbols are not affected and are received and handled normally.
		1b	Port is enabled to respond to any packet
20	ERROR_CHECK_DISABLE		This bit disables all RapidIO transmission error checking. Device behavior when error checking and recovery is disabled and an error condition occurs is undefined.
		0b	Error checking and recovery is enabled
			Error checking and recovery is disabled 1b
19			Multicast-event control symbols to this port (multiple port devices only)
18-4	Reserved		Reserved
3	STOP_PORT_FLD_ENC_ENABLE		When set, this bit causes the port to set the Port Error bit in the Port *n* Error and Status CSR and stop attempting to send packets to the connected device when the Output Failed-encountered bit is set. Packets are discarded if the Drop Packet Enable bit is set. When cleared, the port continues to attempt to transmit packets to the connected device if the Output Failed-encountered bit is set.
2	DROP_PACKET_ENABLE		When set, this bit allows the output port to drop packets that are acknowledged with a packet-not-accepted control symbol when the error failed threshold is exceeded. If the port "heals", and the current error rate falls below the failed threshold, the output no longer drops packets (switch devices only). When cleared, the output port continues to try to transmit packets that have been rejected due to transmission errors
1	PORT_LOCKOUT		Port Lockout: When cleared, this port is enabled to issue any packets. When set, this port is stopped and is not enabled to issue or receive any packets; the input port can still follow the training procedure and can still send and respond to link-requests; all received packets return packet-not-accepted control symbols to force an error condition to be signaled by the sending device.
0	PORT_TYPE		Port type. This indicates the port type, parallel or serial (read only).
		0b	Parallel port
		1b	Serial port

5.86 Error Reporting Block Header (ERR_RPT_BH)

Figure 142. Error Reporting Block Header (ERR_RPT_BH)

31-16
EF_PTR
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
EF_ID
R-0x0007

LEGEND: R = Read only; -*n* = value after reset

Table 116. Error Reporting Block Header (ERR_RPT_BH) Field Descriptions

Bit	Field	Value	Description
31-16	EF_PTR		Hard wired pointer to the next block in the data structure. NONE EXISTS
15-0	EF_ID		Hard wired Extended Features ID

5.87 Logical/Transport Layer Error Detect CSR (ERR_DET)

Figure 143. Logical/Transport Layer Error Detect CSR (ERR_DET)

31	30	29	28	27	26	25	24	23	22	21-16
IO_ERR_RSPNS	MSG_ERR_RSPNS	GSM_ERR_RSPNS	ERR_MSG_FORMAT	ILL_TRANS_DECODE	ILL_TRANS_TRGT_ERR	MSG_REQ_TIMEOUT	PKT_RSPNS_TIMEOUT	UNSOLICITED_RSPNS	UNSUPPORTED_TRANS	Reserved
RC-0x00	RC-0x00	R-0x00	RC-0x00	RC-0x00	R-0x00	RC-0x00	RC-0x00	RC-0x00	RC-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7	6	5-0
Reserved	RX_CPPI_SECURITY	RX_IO_DMA_ACCESS	Reserved
R-0x00	RC-0x00	RC-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 117. Logical/Transport Layer Error Detect CSR (ERR_DET) Field Descriptions

Bit	Field	Value	Description
31	IO_ERR_RSPNS		Received a response of ERROR for an IO Logical Layer Request (endpoint device only). Write 0 to clear.
30	MSG_ERR_RSPNS		Received a response of ERROR for an MSG Logical Layer Request (endpoint device only). Write 0 to clear.
29	GSM_ERR_RSPNS		Received a response of ERROR for a GSM Logical Layer Request (endpoint device only). NOT SUPPORTED _ GSM Logical Layer Not Supported.
28	ERR_MSG_FORMAT		Received MESSAGE packet data payload with an invalid size or segment (MSG logical) (endpoint device only). Write 0 to clear.
27	ILL_TRANS_DECODE		Received illegal fields in the request/response packet for a supported transaction (IO/MSG/GSM logical) (switch or endpoint device). Write 0 to clear.
26	ILL_TRANS_TRGT_ERR		Received a packet that contained a destination ID that is not defined for this endpoint. NOT SUPPORTED _ PACKET DESTROYED BEFORE REACHING LOGICAL LAYER.
25	MSG_REQ_TIMEOUT		A required message request has not been received within the specified time-out interval (MSG logical) (endpoint device only). Write 0 to clear.
24	PKT_RSPNS_TIMEOUT		A required response has not been received within the specified timeout interval (IO/MSG/GSM logical) (endpoint device only). Write 0 to clear.
23	UNSOLICITED_RSPNS		An unsolicited/unexpected Response packet was received (IO/MSG/GSM logical; only Maintenance response for switches) (switch or endpoint device). Write 0 to clear.
22	UNSUPPORTED_TRANS		A transaction is received that is not supported in the Destination Operations CAR (IO/MSG/GSM logical; only Maintenance port-write for switches) (switch or endpoint device). Write 0 to clear
21-8	Reserved		Reserved
7	RX_CPPI_SECURITY		Access to one of the RX queues was blocked. Write 0 to clear.
6	RX_IO_DMA_ACCESS		DMA access to MAU was blocked. Write 0 to clear.
5-0	Reserved		Reserved

5.88 Logical/Transport Layer Error Enable CSR (ERR_EN)

Figure 144. Logical/Transport Layer Error Enable CSR (ERR_EN)

31	30	29	28	27	26	25	24	23	22	21-16
IO_ERR_RESP_ENABLE	MSG_ERR_RESP_ENABLE	GSM_ERR_RESP_ENABLE	ERR_MSG_FORMAT_ENABLE	ILL_TRANS_DECODE_ENABLE	ILL_TRANS_TARGET_ERR_ENABLE	MSG_REQ_TIMEOUT_ENABLE	PKT_RESP_TIMEOUT_ENABLE	UNSOLICITED_RESP_ENABLE	UNSUPPORTED_TRANS_ENABLE	Reserved
RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7	6	5-0
Reserved	RX_CPPI_SECURITY	RX_IO_SECURITY	Reserved
R-0x00	RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 118. Logical/Transport Layer Error Enable CSR (ERR_EN) Field Descriptions

Bit	Field	Value	Description
31	IO_ERR_RESP_ENABLE		Enable reporting of an IO error response (endpoint device only). Save and lock original request transaction capture information in all Logical/Transport Layer Capture CSRs.
30	MSG_ERR_RESP_ENABLE		Enable reporting of a Message error response. Save and lock transaction capture information in all Logical/Transport Layer Capture CSRs. (endpoint device only)
29	GSM_ERR_RESP_ENABLE		Enable reporting of a GSM error response. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock transaction capture information in all Logical/Transport Layer Device ID and Control CSRs. (endpoint device only)
28	ERR_MSG_FORMAT_ENABLE		Enable reporting of a message format error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (endpoint device only)
27	ILL_TRANS_DECODE_ENABLE		Enable reporting of an illegal transaction decode error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
26	ILL_TRANS_TARGET_ERR_ENABLE		Enable reporting of an illegal transaction target error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (endpoint device only) NOT APPLICABLE – PACKET DESTROYED BEFORE REACHING LOGICAL LAYER.
25	MSG_REQ_TIMEOUT_ENABLE		Enable reporting of a Message Request time-out error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs for the last Message request segment packet received. (endpoint device only)
24	PKT_RESP_TIMEOUT_ENABLE		Enable reporting of a packet response time-out error. Save and lock original request address in Logical/Transport Layer Address Capture CSRs. Save and lock original request Destination ID in Logical/Transport Layer Device ID Capture CSR. (endpoint device only)
23	UNSOLICITED_RESP_ENABLE		Enable reporting of an unsolicited response error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
22	UNSUPPORTED_TRANS_ENABLE		Enable reporting of an unsupported transaction error. Save and lock transaction capture information in Logical/Transport Layer Device ID and Control Capture CSRs. (switch or end-point device)
21-8	Reserved		Reserved
7	RX_CPPI_SECURITY		Enable reporting of attempt at unauthorized access to a RX queue. Save and Lock capture information in appropriate Logical/Transport Layer Capture CSRs.
6	RX_IO_SECURITY		Enable reporting of attempt at unauthorized memory location access. Save and Lock capture information in appropriate Logical/Transport Layer Capture CSRs.
5-0	Reserved		Reserved

5.89 Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT)

Figure 145. Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT)

31-16
ADDRESS_63_32
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
ADDRESS_63_32
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 119. Logical/Transport Layer High Address Capture CSR (H_ADDR_CAPT) Field Descriptions

Bit	Field	Value	Description
31-0	ADDRESS_63_32		Most Significant 32 bits of the address associated with the error (only valid for devices supporting 66 and 50 bit addresses)

5.90 Logical/Transport Layer Address Capture CSR (ADDR_CAPT)

Figure 146. Logical/Transport Layer Address Capture CSR (ADDR_CAPT)

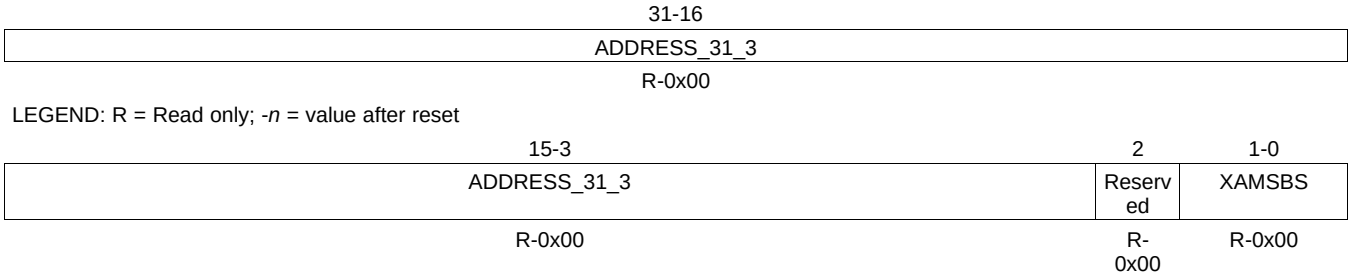


Table 120. Logical/Transport Layer Address Capture CSR (ADDR_CAPT) Field Descriptions

Bit	Field	Value	Description
31-3	ADDRESS_31_3		Least Significant 29 bits of the address associated with the error
2	Reserved		Reserved
1-0	XAMSBS		Extended address bits of the address associated with the error

5.91 Logical/Transport Layer Device ID Capture CSR (ID_CAPT)

Figure 147. Logical/Transport Layer Device ID Capture CSR (ID_CAPT)

31-24	23-16
MSB_DESTID	DESTID
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
MSB_SOURCEID	SOURCEID
R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 121. Logical/Transport Layer Device ID Capture CSR (ID_CAPT) Field Descriptions

Bit	Field	Value	Description
31-24	MSB_DESTID		Most significant byte of the destinationID associated with the error (large transport systems only)
23-16	DESTID		The destinationID associated with the error
15-8	MSB_SOURCEID		Most significant byte of the source ID associated with the error (large transport systems only)
7-0	SOURCEID		The sourceID associated with the error

5.92 Logical/Transport Layer Control Capture CSR (CTRL_CAPT)

Figure 148. Logical/Transport Layer Control Capture CSR (CTRL_CAPT)

31-28	27-24	23-16
FTYPE	TTYPE	MSGINFO
R-0x00	R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
IMP_SPECIFIC
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 122. Logical/Transport Layer Control Capture CSR (CTRL_CAPT) Field Descriptions

Bit	Field	Value	Description
31-28	FTYPE		Format type associated with the error
27-24	TTYPE		Transaction type associated with the error
23-16	MSGINFO		Letter, mbox, and msgseg for the last Message request received for the mailbox that had an error (Message errors only)
15-0	IMP_SPECIFIC		Implementation specific information associated with the error

5.93 Port-Write Target Device ID CSR (PW_TGT_ID)

Figure 149. Port-Write Target Device ID CSR (PW_TGT_ID)

31-24	23-16
DEVICEID_MSB	DEVICEID
RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 123. Port-Write Target Device ID CSR (PW_TGT_ID) Field Descriptions

Bit	Field	Value	Description
31-24	DEVICEID_MSB		This is the most significant byte of the port-write target device ID (large transport systems only)
23-16	DEVICEID		This is the port-write target deviceID
15-0	Reserved		Reserved

5.94 Port Error Detect CSR n (SPn_ERR_DET)

Each of the four ports is supported by a register of this type.

Figure 150. Port Error Detect CSR *n* (SP*n*_ERR_DET)

31	30-24	23	22	21	20	19	18	17	16
ERR_IMP_SPECIFIC	Reserved	Invalid	CORRUPT_CNTL_SYM	CNTL_SYM_UNEXPECTED_ACKID	RCVD_PKT_NOT_ACCPT	PKT_UNEXPECTED_ACKID	RCVD_PKT_WITH_BAD_CRC	RCVD_PKT_OVER_276B	Reserved
RC-0x00	R-0x00	R-0x00	RC-0x00	RC-0x00	RC-0x00	RC-0x00	RC-0x00	RC-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-6	5	4	3	2	1	0
Reserved	NON_OUTSTANDING_ACKID	PROTOCOL_ERROR	Reserved	DELINEATION_ERROR	UNSOLICITED_ACK_CNTL_SYM	LINK_TIMEOUT
R-0x00	RC-0x00	RC-0x00	R-0x00	R-0x00	RC-0x00	RC-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 124. Port Error Detect CSR *n* (SP*n*_ERR_DET) Field Descriptions

Bit	Field	Value	Description
31	ERR_IMP_SPECIFIC		An implementation specific error has been detected. It covers illegal field of Maintenance packet error, illegal destination ID, not supported transaction.
30-24	Reserved		Reserved
23	Invalid		Reserved. Received a packet/control symbol with an S-bit parity error (Not Valid for SERIAL)
22	CORRUPT_CNTL_SYM		Received a control symbol with a bad CRC value (serial)
21	CNTL_SYM_UNEXPECTED_ACKID		Received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet-retry). The Capture Registers don't have valid information during this error detection.
20	RCVD_PKT_NOT_ACCPT		Received packet-not-accepted acknowledge control symbol
19	PKT_UNEXPECTED_ACKID		Received packet with unexpected ackID value- out-of-sequence ackID
18	RCVD_PKT_WITH_BAD_CRC		Received packet with a bad CRC value
17	RCVD_PKT_OVER_276B		Received packet which exceeds the maximum allowed size
16-6	Reserved		Reserved
5	NON_OUTSTANDING_ACKID		Link-response received with an ackID that is not outstanding. The Capture Registers don't have valid information during this error detection.
4	PROTOCOL_ERROR		An unexpected packet or control symbol was received
3	Reserved		Reserved
2	DELINEATION_ERROR		Received unaligned /SC/ or /PD/ or undefined code-group (serial). The Capture Registers don't have valid information during this error detection.
1	UNSOLICITED_ACK_CNTL_SYM		An unexpected acknowledge control symbol was received
0	LINK_TIMEOUT		An acknowledge or link-response control symbol is not received within the specified time-out interval. The Capture Registers don't have valid information during this error detection.

5.95 Port Error Rate Enable CSR n (SPn_RATE_EN)

Each of the four ports is supported by a register of this type.

Figure 151. Port Error Rate Enable CSR *n* (SP*n*_RATE_EN)

31	30-24	23	22	21	20	19	18	17	16
EN_IMP_SPECIFIC	Reserved	Reserved	CORRUPT_CNTL_SYM_ENABLE	CNTL_SYM_UNEXPECTED_ACKID_EN	RCVED_PKT_NOT_ACCPT_EN	PKT_UNEXPECTED_ACKID_EN	RCVED_PKT_WITH_BAD_CRC_EN	RCVED_PKT_OVER_276B_EN	Reserved
RW-0x00	R-0x00	R-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-6	5	4	3	2	1	0
Reserved	NON_OUTSTANDING_ACKID_EN	PROTOCOL_ERROR_EN	Reserved	DELINEATION_ERROR_EN	UNSOLICITED_ACK_CNTL_SYM_EN	LINK_TIMEOUT_EN
R-0x00	RW-0x00	RW-0x00	R-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 125. Port Error Rate Enable CSR *n* (SP*n*_RATE_EN) Field Descriptions

Bit	Field	Value	Description
31	EN_IMP_SPECIFIC		Enable error rate counting of implementation specific errors
30-24	Reserved		Reserved
23	Reserved		Reserved
22	CORRUPT_CNTL_SYM_ENABLE		Enable error rate counting of a corrupt control symbol
21	CNTL_SYM_UNEXPECTED_ACKID_EN		Enable error rate counting of an acknowledge control symbol with an unexpected ackID
20	RCVED_PKT_NOT_ACCPT_EN		Enable error rate counting of received packet-not-accepted control symbols
19	PKT_UNEXPECTED_ACKID_EN		Enable error rate counting of packet with unexpected ackID value- out-of-sequence ackID
18	RCVED_PKT_WITH_BAD_CRC_EN		Enable error rate counting of packet with a bad CRC value
17	RCVED_PKT_OVER_276B_EN		Enable error rate counting of packet which exceeds the maximum allowed size
16-6	Reserved		Reserved
5	NON_OUTSTANDING_ACKID_EN		Enable error rate counting of link-responses received with an ackID that is not outstanding
4	PROTOCOL_ERROR_EN		Enable error rate counting of protocol errors
3	Reserved		Reserved
2	DELINEATION_ERROR_EN		Enable error rate counting of delineation errors
1	UNSOLICITED_ACK_CNTL_SYM_EN		Enable error rate counting of unsolicited acknowledge control symbol errors
0	LINK_TIMEOUT_EN		Enable error rate counting of link time-out errors

5.96 Port n Attributes Error Capture CSR 0 (SPn_ERR_ATTR_CAPT_DBG0)

Each of the four ports is supported by a register of this type.

Figure 152. Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0)

31-30	29	28-24	23-16
INFO_TYPE	Reserved	ERROR_TYPE	IMP_SPECIFIC
R-0x00	R-0x00	R-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-4	3-1	0
IMP_SPECIFIC	Reserved	CAPTURE_VALID_INFO
R-0x00	R-0x00	RC-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 126. Port *n* Attributes Error Capture CSR 0 (SP*n*_ERR_ATTR_CAPT_DBG0) Field Descriptions

Bit	Field	Value	Description
31-30	INFO_TYPE		Type of information logged.
		00	Packet
		01	Control symbol (only error capture register 0 is valid)
		10	Implementation specific (capture register contents are implementation specific)
		11	Undefined (S-bit error), capture as if a packet (parallel physical layer only)
29	Reserved		Reserved
28-24	ERROR_TYPE		Encoded value of captured error bit in the Port *n* Error Detect Register
23-4	IMP_SPECIFIC		Implementation Dependent Error Information
3-1	Reserved		Reserved
0	CAPTURE_VALID_INFO		This bit is set by hardware to indicate that the Packet/control symbol capture registers contain valid information. For control symbols, only capture register 0 will contain meaningful information. A software write of 0 clears this bit and subsequently unlocks all capture registers of port *n*.

5.97 Port n Packet/Control Symbol Error Capture CSR 1 (SPn_ERR_CAPT_DBG1)

Each of the four ports is supported by a register of this type.

Figure 153. Port *n* Packet/Control Symbol Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1)

31-16
CAPTURE0
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
CAPTURE0
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 127. Port *n* Packet/Control Symbol Error Capture CSR 1 (SP*n*_ERR_CAPT_DBG1) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE0		Control Character and Control Symbol or 0 to 3 Bytes of Packet Header

5.98 Port n Packet/Control Symbol Error Capture CSR 2 (SPn_ERR_CAPT_DBG2)

Each of the four ports is supported by a register of this type.

Figure 154. Port *n* Packet/Control Symbol Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2)

31-16
CAPTURE1
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
CAPTURE1
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 128. Port *n* Packet/Control Symbol Error Capture CSR 2 (SP*n*_ERR_CAPT_DBG2) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE1		4 to 7 Bytes of Packet Header

5.99 Port n Packet/Control Symbol Error Capture CSR 3 (SPn_ERR_CAPT_DBG3)

Each of the four ports is supported by a register of this type.

Figure 155. Port *n* Packet/Control Symbol Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3)

31-16
CAPTURE2
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
CAPTURE2
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 129. Port *n* Packet/Control Symbol Error Capture CSR 3 (SP*n*_ERR_CAPT_DBG3) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE2		8 to 11 Bytes of Packet Header

5.100 Port n Packet/Control Symbol Error Capture CSR 4 (SPn_ERR_CAPT_DBG4)

Each of the four ports is supported by a register of this type.

Figure 156. Port *n* Packet/Control Symbol Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4)

31-16
CAPTURE3
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
CAPTURE3
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 130. Port *n* Packet/Control Symbol Error Capture CSR 4 (SP*n*_ERR_CAPT_DBG4) Field Descriptions

Bit	Field	Value	Description
31-0	CAPTURE3		12 to 15 Bytes of Packet Header

5.101 Port Error Rate CSR n (SPn_ERR_RATE)

Each of the four ports is supported by a register of this type.

Figure 157. Port Error Rate CSR *n* (SP*n*_ERR_RATE)

31-24	23-18	17-16
ERROR_RATE_BIAS	Reserved	ERROR_RATE_RECOVERY
RW-0xFF	R-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
PEAK_ERROR_RATE	ERROR_RATE_COUNTER
RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 131. Port Error Rate CSR *n* (SP*n*_ERR_RATE) Field Descriptions

Bit	Field	Value	Description
31-24	ERROR_RATE_BIAS		These bits provide the error rate bias value.
		0x00	Do not decrement the error rate counter
		0x01	Decrement every 1ms (nominal)
		0x03	Decrement every 10ms (nominal)
		0x07	Decrement every 100ms
		0x0F	Decrement every 1s (nominal)
		0x1F	Decrement every 10s (nominal)
		0x3F	Decrement every 100s (nominal)
		0x7F	Decrement every 1000s (nominal)
		0xFF	Decrement every 10000s (nominal)
			Other values are reserved
23-18	Reserved		Reserved
17-16	ERROR_RATE_RECOVERY		These bits limit the incrementing of the error rate counter above the failed threshold trigger.
		00b	Only count 2 errors above
		01b	Only count 4 errors above
		10b	Only count 16 errors above
		11b	Do not limit incrementing the error rate count
15-8	PEAK_ERROR_RATE		This field contains the peak value attained by the error rate counter.
7-0	ERROR_RATE_COUNTER		These bits maintain a count of the number of transmission errors that have occurred. If this value equals the value contained in the error rate threshold trigger register, then an error will be reported.

5.102 Port Error Rate Threshold CSR n (SPn_ERR_THRESH)

Each of the four ports is supported by a register of this type.

Figure 158. Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH)

31-24	23-16
ERROR_RATE_FAILED_THRESHOLD	ERROR_RATE_DEGRADED_THRES
RW-0xFF	RW-0xFF

LEGEND: R = Read only; -*n* = value after reset

15-0
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 132. Port Error Rate Threshold CSR *n* (SP*n*_ERR_THRESH) Field Descriptions

Bit	Field	Value	Description
31-24	ERROR_RATE_FAILED_THRESHOLD		These bits provide the threshold value for reporting an error condition due to a possibly broken link.
		0x00	Disable the error rate register
		0x01	Set the error reporting threshold to 1
		0x02	Set the error reporting threshold to 2
		...	
		0xFF	Set the error reporting threshold to 255
23-16	ERROR_RATE_DEGRADED_THRES		These bits provide the threshold value for reporting an error condition due to a degrading link.
		0x00	Disable the error rate Register
		0x01	Set the error reporting threshold to 1
		0x02	Set the error reporting threshold to 2
		...	
		0xFF	Set the error reporting threshold to 255
15-0	Reserved		Reserved

5.103 Port IP Discovery Timer in 4x mode (SP_IP_DISCOVERY_TIMER)

Figure 159. Port IP Discovery Timer in 4x mode (SP_IP_DISCOVERY_TIMER)

31-28	27-24	23-20	19-16
DISCOVERY_TIMER	Reserved	PW_TIMER	Reserved
RW-0x09	R-0x00	RW-0x08	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 133. Port IP Discovery Timer in 4x mode (SP_IP_DISCOVERY_TIMER) Field Descriptions

Bit	Field	Value	Description
31-28	DISCOVERY_TIMER		Discovery Timer in 4x mode. The discovery-timer allows time for the link partner to enter its DISCOVERY state and if the link partner is supporting 4x mode, for all 4 lanes to be aligned.
		0000b	102.4pS, for debug only
		0001b	0.84ms
		0010b	0.84ms * 2 = 1.68ms
		...	
		1001b	0.84ms * 9= 7.56ms (default)
		...	
		1111b	0.84ms *15= 12.6ms
27-24	Reserved		Reserved
23-20	PW_TIMER		Port-Write Timer. The timer defines a period to repeat sending an error reporting Port-Write request for software assistance. The timer is stopped by software writing to the error detect registers.
		0000b	Disabled. Port-Write is sent once only.
		0001b	107ms-214ms
		0010b	214ms-321ms
		0100b	428ms-535ms
		1000b	856ms-963ms (default)
		1111b	0.82-1.64us (for debug only)
		Other	Reserved
19-0	Reserved		Reserved

5.104 Port IP Mode CSR (SP_IP_MODE)

Figure 160. Port IP Mode CSR (SP_IP_MODE)

31-30	29	28	27	26	25	24-16
SP_MODE	IDLE_ERR_DIS	TX_FIFO_BYPASS	PW_DIS	TGT_ID_DIS	SELF_RST	Reserved
R-0x00	RW-0x00	RW-0x00	RW-0x00	R-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-6	5	4	3	2	1	0
Reserved	MLTC_EN	MLTC_IRQ	RST_EN	RST_CS	PW_EN	PW_IRQ
R-0x00	RW-0x00	RC-0x00	RW-0x00	RC-0x00	RW-0x00	RC-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 134. Port IP Mode CSR (SP_IP_MODE) Field Descriptions

Bit	Field	Value	Description
31-30	SP_MODE		SRIO Port IP Mode of operation.
		00	1x/4x LP-Serial RapidIO Specification
		01	4 ports (1x mode each)
		10	Reserved
		11	Reserved
29	IDLE_ERR_DIS		IDLE Error checking disable.
		0	Error checking enabled (default), only |K|, |A| and |R| characters are available. If input receives any other characters in idle sequence, it should enter the Input-Error-stopped state.
		1	Error checking disabled, all not idle or invalid characters during idle sequence are ignored
28	TX_FIFO_BYPASS		Transmit FIFO
		0	The TX_FIFO is operational (Default)
		1	The TX_FIFO is bypassed. The txbclk and the sys_clk must be locked during operation, but the phase variation up to 1 clock cycle is allowable. The 4 deep FIFO is used to accommodate the phase difference.
27	PW_DIS		Port-Write Disable.
		0	Enable Port-Write Error reporting (default)
		1	Disable Port-Write Error reporting
26	TGT_ID_DIS		Destination ID Decode Disable- Definition of packet acceptance by the physical layer.
		0	Packet accepted if DestID = Base ID. When DestID is not equal to Base ID, the packet is ignored; i.e., it is accepted by RapidIO port but is not forwarded to logical layer.
		1	Packet accepted with any DestID and forwarded to the logical layer.
25	SELF_RST		Self reset enable, when 4 Link-Request Reset Control Symbols are accepted.
		0	Self reset interrupt disabled (default), interrupt signal is asserted
		1	Self reset interrupt enabled, the reset signal is asserted by the SRIO_TE reset controller. The SRIO_TE configuration registers are set to the default value after reset and loose a boot initialization values.
24-6	Reserved		Reserved
5	MLTC_EN		Multicast-Event Interrupt Enable. If enabled, the interrupt signal is High when the Multicast-Event control symbol is received by any port.
		0b	Multicast interrupt disable
		1b	Multicast interrupt enable
4	MLTC_IRQ		Multicast-Event Interrupt Status bit. It is High when the Multicast-Event control symbol is received by any port. Once set, it remains set until written with logic 1 to clear. The mltc_irq output signal is driven by this bit.

Table 134. Port IP Mode CSR (SP_IP_MODE) Field Descriptions (continued)

Bit	Field	Value	Description
3	RST_EN		Reset Interrupt Enable. If enabled, the interrupt signal is High when the 4 reset control symbols are received in a sequence
		0b	Reset interrupt disable
		1b	Reset interrupt enable
2	RST_CS		Reset received status bit. It is set when 4 reset control symbols are received in a sequence. Once set, it remains set until written with logic 1 to clear. The rst_irq output signal is driven by this bit.
1	PW_EN		Port-Write-In Interrupt Enable. If enabled, the interrupt signal is High when the Port-Write-In request is received
		0b	Port-Write-In interrupt disable
		1b	Port-Write-In interrupt enable
0	PW_IRQ		Port-Write-In Request interrupt is set when the Port-Write-In request is received. The payload is captured. Once set, it remains set until written with logic 1 to clear. The pw_irq output signal is driven by this bit.

5.105 Serial Port IP Prescalar (IP_PRESCAL)

Figure 161. Serial Port IP Prescalar (IP_PRESCAL)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
Reserved	PRESCALE
R-0x00	RW-0x0F

LEGEND: R = Read only; -*n* = value after reset

Table 135. Serial Port IP Prescalar (IP_PRESCAL) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved		Reserved
7-0	PRESCALE		For different frequencies of the DMA clock, use the formula: [DMA freq * 16/156.25 _ 1] to get the prescalar value in decimal, where DMA freq is in MHz.
		06h	66.67 MHz
		09h	100MHz
		10h	166.67MHz
		...	
		18h	250MHz
		...	
		21h	333Mhz

5.106 Port-Write-In Capture CSR n (SP_IP_PW_IN_CAPTn)

Each of the four ports is supported by a register of this type.

Figure 162. Port-Write-In Capture CSR *n* (SP_IP_PW_IN_CAPT*n*)

31-16
PW_CAPT*n*
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
PW_CAPT*n*
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 136. Port-Write-In Capture CSR *n* (SP_IP_PW_IN_CAPT*n*) Field Descriptions

Bit	Field	Value	Description
31-0	PW_CAPT		Port-Write payload: word *n*

5.107 Port Reset Option CSR n (SPn_RST_OPT)

Each of the four ports is supported by a register of this type.

Figure 163. Port Reset Option CSR *n* (SP*n*_RST_OPT)

31-16
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7-0
Reserved	PORT_ID
R-0x00	R-Undefined

LEGEND: R = Read only; -*n* = value after reset

Table 137. Port Reset Option CSR *n* (SP*n*_RST_OPT) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved		Reserved
7-0	PORT_ID		Port ID defines unique number for port in Switch. The Port ID is used for Port_Write request. The ID coincides with ISF port of connection. Example: 00_0000_01 _ port 1 (Impl.: IP0, port 1) 00_0001_11 _ port 7 (Impl.: IP1, port 3).

5.108 Port Control Independent Register n (SPn_CTL_INDEP)

Each of the four ports is supported by a register of this type.

Figure 164. Port Control Independent Register *n* (SP*n*_CTL_INDEP)

31	30	29	28-27	26	25-24	23	22	21	20	19-18	17	16
Reserved	TX_FLW	SOFT_REC	Reserved	FORCE_REINIT	TRANS_MODE	DEBUG	SEND_DBG_PKT	ILL_TRANS_EN	ILL_TRANS_ERR	Reserved	MAX_RETRY_EN	MAX_RETRY_ERR
R-0x00	RW-0x00	RW-0x00	R-0x00	W-0x00	RW-0x01	RW-0x00	RW-0x00	RW-0x00	RC-0x00	R-0x00	RW-0x00	RC-0x00

LEGEND: R = Read only; -*n* = value after reset

15-8	7	6	5-0
MAX_RETRY_THR	IRQ_EN	IRQ_ERR	Reserved
RW-0x00	RW-0x00	RC-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 138. Port Control Independent Register *n* (SP*n*_CTL_INDEP) Field Descriptions

Bit	Field	Value	Description
31	Reserved		Reserved
30	TX_FLW		Transmit Link Flow Control enable
		0b	Disables transmit flow control (Enables receive link flow control)
		1b	Enables transmit flow control (not supported)
29	SOFT_REC		Software controlled error recovery.
		0b	Transmission of error recovery sequence is performed by the hardware
		1b	Transmission of error recovery sequence is performed by the software. By default the transmission error recovery sequence is performed by the hardware. If this bit is set, the hardware recovery is disabled and the hardware transmission logic must wait until software has written the register Port *n* Local ackID Status CSR.
28-27	Reserved		Reserved
26	FORCE_REINIT		Force reinitialization process. In 4x mode this bit affects all 4 lanes. This bit is write only, read always "0".
25-24	TRANS_MODE		Describes the transfer mode for each port.
		00	Reserved (Cut-Through Mode)
		01	Store & Forward Mode
		1x	Reserved
23	DEBUG		Mode of operation.
		0b	Normal mode
		1b	Debug mode. The debug mode unlocks capture registers for write and enable debug packet generator feature.
22	SEND_DBG_PKT		Send debug packet. Write 1 force to send debug packet. This bit is set by software and cleared after debug packet is sent. Writes when the bit is set are ignored. Debug mode only.
21	ILL_TRANS_EN		Illegal Transfer Error reporting Enable. If enabled, the Port-Write and interrupt are reported errors.
		0b	Disables Illegal Transfer Error reporting
		1b	Enables Illegal Transfer Error reporting
20	ILL_TRANS_ERR		Illegal Transfer Error. It is set to 1 as following: • Received transaction has reserved tt field • Reserved field of Maintenance transaction type • DestID is not defined in look-up table Once set, it remains set until written with logic 1 to clear. This bit also is cleared by writing all 0s to the register SP0_ERR_DET. This error is also reported in registers SP0_ERR_DET and ERR_DET.
19-18	Reserved		Reserved

Table 138. Port Control Independent Register *n* (SP*n*_CTL_INDEP) Field Descriptions (continued)

Bit	Field	Value	Description
17	MAX_RETRY_EN		Max_retry_error report enable. If enabled, the Port-Write and interrupt are reported as errors.
		1b	Max retry error report enable
		0b	Max retry error report disable
16	MAX_RETRY_ERR		Max_retry_error bit is set when max_retry_cnt is equal to max_retry_threshold. The Port-Write request and interrupt are generated if enabled. This bit is ignored if max_retry threshold is 00. Once set, it remains set until written with logic 1 to clear. This bit also is cleared by writing all 0s to the register SP0_ERR_DET. This error also reported in register SP0_ERR_DET.
15-8	MAX_RETRY_THR		Maximum Retry Threshold Trigger. These bits provide the threshold value for reporting an error condition due to possibly broken partner behavior.
		00	Disable the max_retry_error reporting
		01	Set the max_retry_threshold to 1
		02	Set the max_retry_threshold to 2
		...	
		FF	Set the max_retry_threshold to 255
7	IRQ_EN		Interrupt error report Enable. If enabled, the interrupt signal is High when the IRQ_ERR is set to 1.
		1b	Interrupt error report enable
		0b	Interrupt error report disable
6	IRQ_ERR		The Interrupt Error Status bit. It is set to one if an error occurs and there is a Port-Write condition. Once set, remains set until written with logic 1 to clear.
5-0	Reserved		Reserved

5.109 Port Silence Timer n (SPn_SILENCE_TIMER)

Each of the four ports is supported by a register of this type.

Figure 165. Port Silence Timer *n* (SP*n*_SILENCE_TIMER)

31-28	27-16
SILENCE_TIMER	Reserved
RW-0x0B	R-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
Reserved
R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 139. Port Silence Timer *n* (SP*n*_SILENCE_TIMER) Field Descriptions

Bit	Field	Value	Description
31-28	SILENCE_TIMER		Silence timer. Defines the time of the port in the SILENT state.
		0000b	64ns for debug
		0001b	13.1us
		0010b	13.1us * 2 = 26.2us
		...	
		1011b	13.1us * 11 = 144.1us default
		...	
		1111b	13.1us *15= 196.5us
27-0	Reserved		Reserved

5.110 Port Multicast-Event Control Symbol Request Register n (SPn_MULT_EVNT_CS)

Each of the four ports is supported by a register of this type.

Figure 166. Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS)

31-16
MULT_EVNT_CS
W-0x00

LEGEND: R = Read only; -*n* = value after reset

15-0
MULT_EVNT_CS
W-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 140. Port Multicast-Event Control Symbol Request Register *n* (SP*n*_MULT_EVNT_CS) Field Descriptions

Bit	Field	Value	Description
31-0	MULT_EVNT_CS		Write to send Control Symbol, data is ignored. read- 0x000000.

5.111 Port Control Symbol Transmit n (SPn_CS_TX)

Each of the four ports is supported by a register of this type.

Figure 167. Port Control Symbol Transmit *n* (SP*n*_CS_TX)

31-29	28-24	23-19	18-16
STYPE_0	PAR_0	PAR_1	STYPE_1
RW-0x00	RW-0x00	RW-0x00	RW-0x00

LEGEND: R = Read only; -*n* = value after reset

15-13	12	11-0
CMD	CS_EMB	Reserved
RW-0x00	RW-0x00	R-0x00

LEGEND: R = Read only; -*n* = value after reset

Table 141. Port Control Symbol Transmit *n* (SP*n*_CS_TX) Field Descriptions

Bit	Field	Value	Description
31-29	STYPE_0		Encoding for control symbol that makes use of parameters PAR_0 and PAR_1.
28-24	PAR_0		Used in conjunction with stype0 encoding.
23-19	PAR_1		Used in conjunction with stype0 encoding.
18-16	STYPE_1		Encoding for control symbol that makes use of parameter CMD.
15-13	CMD		Used in conjunction with stype1 encoding to define the link maintenance commands.
12	CS_EMB		When set, forces the outbound flow to insert control symbol into packet. Used in debug mode.
11-0	Reserved		Reserved

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265